(19) BUNDESREPUBLIK DEUTSCHLAND

DEUTSCHES PATENTAMT

(12) **Offenlegungsschrift**

(10) **DE 42 05 524 A 1**

(51) Int. Cl.:
G 05 B 19/05
B 65 B 57/00
B 65 C 9/40

(21) Aktenzeichen: P 42 05 524.5
(22) Anmeldetag: 24. 2. 92
(43) Offenlegungstag: 27. 8. 92

DE 42 05 524 A 1

(30) Unionspriorität: (32) (33) (31)
14.11.91 EP 91118483.5

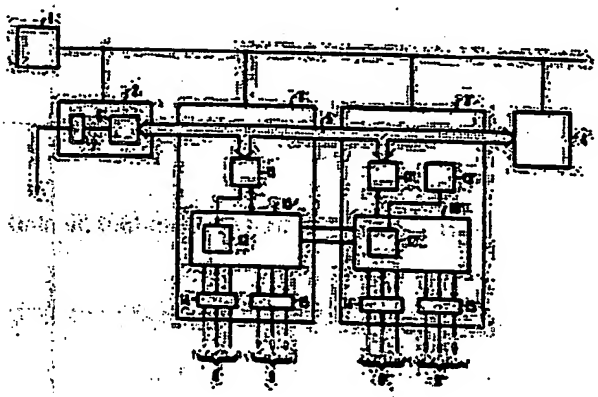
(30) Innere Priorität: (32) (33) (31)
22.02.91 DE 41 05 678.7

(71) Anmelder:
Siemens AG, 8000 München, DE

(72) Erfinder:
Bock, Günther, Dipl.-Ing., 8450 Amberg, DE; Mecht, Helmut, Dipl.-Ing., 8457 Kümmersbruck, DE; Wombacher, Christof, Dipl.-Ing. (FH), 8450 Amberg, DE; Prechtl, Manfred, Dipl.-Ing. (Univ.), 8470 Nabburg, DE; Langemann, Andre, Dipl.-Ing. (FH), 8459 Edelsfeld, DE

(54) Speicherprogrammierbare Steuerung

(57) Es wird eine neue speicherprogrammierbare Steuerung, insbesondere für Verpackungs- und Etikettiermaschinen mit mehreren Ein- (8) und Ausgängen (9) zum Anschließen von Prozeßführungselementen, z. B. Sensoren oder Stellgliedern, vorgeschlagen, die mindestens einen Logikbaustein (10) mit einer internen Verschaltung aufweist, über die mindestens ein Ausgang mit seinem korrespondierenden Eingang (E0) verbunden ist. Ferner wird ein Programmierverfahren für einen derartigen Logikbaustein (10) vorgestellt.



DE 42 05 524 A 1

Beschreibung

Die Erfindung betrifft eine speicherprogrammierbare Steuerung, insbesondere für Verpackungs- und Etikettiermaschinen, mit mehreren Ein- und Ausgängen zum Anschließen von Prozeßführungselementen, z. B. Sensoren oder Stellgliedern, sowie ein Verfahren zum Betreiben einer speicherprogrammierbaren Steuerung und ein Verfahren zum rechnergesteuerten internen elektrischen Verbinden eines umfeldprogrammierbaren Logikfeldes.

Früher wurden Maschinensteuerungen in Schütz-Technik aufgebaut. Schütz-Schaltungen arbeiten zwar parallel und sind daher schnell, sie sind jedoch störanfällig, kompliziert und nur umständlich aufzubauen bzw. anzupassen. Inzwischen sind speicherprogrammierbare Steuerungen weit verbreitet. Sie arbeiten sequentiell und sind erheblich einfacher aufgebaut und zu programmieren. Aber auch moderne speicherprogrammierbare Steuerungen sind wegen ihrer sequentiellen Arbeitsweise oftmals nicht schnell genug, z. B. für die Steuerung von Verpackungs- oder Etikettiermaschinen. Steuerungen für diese Maschinen werden in der Regel auch heute noch auf der Basis zu verdrahtender Logikelemente aufgebaut. Dadurch bieten diese Steuerungen zwar eine hohe Verarbeitungsgeschwindigkeit, jedoch ist das Verdrahten der Logikelemente sehr umständlich und fehleranfällig.

Aufgabe der vorliegenden Erfindung ist es, eine speicherprogrammierbare Steuerung zur Verfügung zu stellen, mit der es möglich ist, extrem schnelle Steuerungsvorgänge zu bewältigen. Weiterhin soll ein Verfahren angegeben werden, das es ermöglicht, eine derartige speicherprogrammierbare Steuerung, die einen Logikbaustein mit einem umfeldprogrammierbaren Logikfeld enthält, schnell und einfach zu programmieren.

Die erste Aufgabe wird dadurch gelöst, daß die Steuerung mindestens einen Logikbaustein mit einer internen Verschaltung aufweist, über die mindestens ein Ausgang mit einem korrespondierenden Eingang verbunden ist. Dadurch wird ständig das Ausgangssignal dieses Ausgangs an den Wert des korrespondierenden Eingangs angepaßt, so daß eine ansonsten nötige, aufwendige Alarmreaktion bei diesem Eingang entfallen kann.

Mit Vorteil ist der Logikbaustein parallelarbeitend ausgebildet, so daß mehrere Ein- und Ausgänge miteinander verbindbar sind. Dadurch wird erreicht, daß die Ausgangssignale dieser Ausgänge keinen durch die Verarbeitungszeit verursachten Schwebungen und Schwankungen unterliegen, sondern stabil reproduzierbar sind.

Wenn die interne Verschaltung des Logikbausteins programmierbar und insbesondere auch reprogrammierbar ist, ist die Steuerung leicht an geänderte Anforderungen anpaßbar.

Die Programmierung des Logikbausteins ist dabei besonders einfach, wenn der Logikbaustein einen — vorzugsweise statischen — Speicher zum Speichern der Bedingungen aufweist, die seine interne Verschaltung festlegen. Der Logikbaustein kann insbesondere ein umfeldprogrammierbares Logikfeld (FPGA) sein.

Wenn die Steuerung modular aufgebaut ist, ist der Logikbaustein vorteilhaft in einer Ein-/Ausgabe-Baugruppe angeordnet, weil dann der Systembus der Steuerung nicht für den Datentransfer benutzt werden muß. In diesem Fall ist es weiterhin von Vorteil, wenn der Logikbaustein direkt auf der Baugruppe programmiert werden kann, z. B. über eine Schnittstelle zum Anschluß eines Daten-Ein-/Ausgabegeräts, wobei die Schnittstelle direkt oder indirekt auf den Systembus oder aber auch auf den Logikbaustein selbst wirkt, oder über einen zusteckbaren Anwenderspeicher, der die interne Verschaltung des Logikbausteins festlegt.

Die Steuerung arbeitet derart, daß mindestens ein Eingangssignal in einen Logikbaustein eingelesen und dort verarbeitet wird, so daß vom Logikbaustein ein mit dem Eingangssignal korrespondierendes Ausgangssignal ausgebbar ist.

Die zweite Aufgabe wird durch folgende Verfahrensschritte gelöst:

- aus einem vorgegebenen funktionalen Gesamtverhalten, z. B. aufgrund eines vorgegebenen funktionalen Schaltplans, insbesondere eines Funktionsplans für eine speicherprogrammierbare Steuerung, werden interne elektrische Konfigurationen, also Verbindungen und gegebenenfalls auch Logikfunktionen des Logikfeldes, bestimmt, welche das vorgegebene funktionale Gesamtverhalten realisieren und
- die so bestimmten internen elektrischen Konfiguration, also Verbindungen und gegebenenfalls auch Logikfunktionen, werden dem Logikfeld eingeprägt,
- wobei unabhängig von dem vorgegebenen funktionalen Gesamtverhalten beim Festlegen der internen elektrischen Verbindungen ein Teil der im Prinzip frei festlegbaren internen elektrischen Verbindungen fest vorgegeben wird.

Wenn die Logikblöcke durch den fest vorgegebenen Teil der internen elektrischen Verbindungen in Gruppen aufgeteilt werden, die zumindest teilweise gleiche Konfigurationen aufweisen, werden regelmäßige Strukturen erzeugt, so daß das Logikfeld quasi in kleinere Einheiten, nämlich die Gruppen, zerlegt wird. Dadurch ist es nämlich möglich, das vorgegebene funktionale Gesamtverhalten in Teilfunktionen zu zerlegen, die zumindest teilweise in einer Gruppe von Logikblöcken realisierbar sind.

Für Standard-Teilfunktionen, insbesondere komplexe Standard-Teilfunktionen, sind dabei interne elektrische Standard-Konfigurationen, also Verbindungen und gegebenenfalls auch Logikfunktionen, vorgebbar, wobei im Einzelfall die Standard-Teilfunktionen auch durch mehr als eine Gruppe von Logikblöcken realisierbar sein können.

Wenn Teilfunktionen und Standard-Teilfunktionen im Einzelfall sehr einfach sind, können gegebenenfalls mehrere von ihnen zusammengefaßt werden, sofern auch die Zusammenfassung in einer Gruppe von Logikblöcken realisierbar ist.

Die Programmierung des Logikfeldes erfolgt dann dadurch, daß

- die Teilfunktionen und/oder die Standard-Teilfunktionen und/ oder die Zusammenfassung den Gruppen von Logikblöcken zugeordnet werden,
- unter Berücksichtigung der fest vorgegebenen internen elektrischen Verbindungen die internen elektrischen Verbindungen ermittelt werden, die die Gruppen von Logikblöcken derart miteinander und mit externen Anschlüssen verschalten, daß das vorgegebene funktionale Gesamtverhalten realisiert wird, und
- die so ermittelten internen elektrischen Verbindungen dem Logikfeld eingeprägt werden, vor-

zugsweise zusammen mit den fest vorgegebenen internen elektrischen Verbindungen.

Die Programmierung des Logikfeldes ist für den Anwender besonders einfach, wenn das funktionale Gesamtverhalten in einer Programmiersprache für speicherprogrammierbare Steuerungen vorgegeben wird, insbesondere in einer graphischen Programmiersprache.

Weitere Vorteile und Einzelheiten ergeben sich aus der nachfolgenden Beschreibung eines Ausführungsbeispiels anhand der Zeichnungen und in Verbindung mit den weiteren Unteransprüchen. Es zeigt

Fig. 1 mehrere Baugruppen einer modular aufgebauten speicherprogrammierbaren Steuerung,

Fig. 2 den inneren Aufbau einer Ein-/Ausgabe-Baugruppe,

Fig. 3 die Verbindungen zwischen Ein- und Ausgängen,

Fig. 4 den konstruktiven Aufbau einer Ein-/Ausgabe-Baugruppe,

Fig. 5 die innere Struktur eines umfeldprogrammierbaren Logikfeldes,

Fig. 6 den Aufbau eines Logikblocks,

Fig. 7 eine beispielhaft zu lösende Problemstellung in Form einer Ampelanlage,

Fig. 8 die zugehörige schaltechnische Realisierung der Ampelanlagensteuerung,

Fig. 9 die Vorabfestlegung der internen elektrischen Verbindungen,

Fig. 10 ein Beispiel einer internen elektrischen Standard-Verbindung,

Fig. 11 die Realisierung des vorgegebenen Gesamtverhaltens im umfeldprogrammierbaren Logikfeld und

Fig. 12 schematisch die Kommunikation zwischen Logikbaustein und Prozessor.

Gemäß Fig. 1 besteht eine modular aufgebaute speicherprogrammierbare Steuerung aus einer Stromversorgung 1, einer Zentraleinheit 2, den Ein-/Ausgabe-Baugruppen 3, 3' sowie weiteren Peripherieeinheiten 4. Die Baugruppen 2, 3, 3', 4 sind dabei über einen Bus 5 miteinander verbunden. Die Zentraleinheit 2 weist mindestens einen Prozessor 6 zum Abarbeiten eines Programmes sowie eine Schnittstelle 7 zum Datenaustausch mit einem Programmiergerät auf.

Wie weiterhin aus Fig. 1 ersichtlich ist, weist die Baugruppe 3 einen Logikbaustein 10 auf, der z. B. ein umfeldprogrammierbares Logikfeld (FPGA) sein kann. Der Logikbaustein 10 ist über den Prozessor 11 mit dem Bus 5 und damit auch mit der Zentraleinheit 2 verbunden. Dadurch ist es möglich, von einem Programmiergerät aus über die Zentraleinheit 2 die interne Verschaltung des Logikbausteins 10 derart zu programmieren, daß die Eingänge 8 über den Logikbaustein 10 gemäß zuvor aus dem abzuarbeitenden Programm abgeleiteten logischen Bedingungen zwischen Eingangs- und Ausgangssignalen mit den Ausgängen 9 verbunden sind. Der Anwender erstellt hierzu mit dem oben erwähnten Programmiergerät zwei Programmteile: Einen zeitunkritischen Teil und einen zeitkritischen Teil. Beide Teile werden vom Programmiergerät an den Prozessor 6 der Zentraleinheit 2 übertragen. Der zeitunkritische Teil wird in der Zentraleinheit 2 abgespeichert und wie bei speicherprogrammierbaren Steuerungen allgemein üblich sequentiell abgearbeitet. Der zeitkritische Teil wird vom Prozessor 6 weiter an die Logikbausteine 10, 10' übertragen und von diesen in eine logische Verschaltung umgesetzt.

Die beiden Programmteile sind völlig unabhängig voneinander. Es ist jedoch über Sonderbefehle möglich, daß der Prozessor 6 und die Logikbausteine 10, 10 Informationen austauschen.

Mit Vorteil werden dabei die logischen Bedingungen, die die Verschaltung des Logikbausteins 10 festlegen, in einen statischen Speicher 12 des Logikbausteins 10 übertragen und die Verschaltung des Logikbausteins 10 aufgrund des Inhalts des Speichers 12 bestimmt.

Die Baugruppe 3' weist ebenfalls einen Logikbaustein 10 mit einem statischen Speicher 12' auf, der Logikbaustein 10' wird jedoch über einen Anwenderspeicher 13' programmiert. Wenn die Verschaltung des Logikbausteins 10' geändert werden soll, muß der Anwenderspeicher 13' ausgetauscht bzw. umprogrammiert werden, da die logischen Bedingungen, die die Verschaltung des Logikbausteins 10' bestimmen, im Anwenderspeicher 13' abgespeichert sind.

Fig. 2 zeigt in etwas geänderter Darstellung den elektrischen Aufbau der Baugruppe 3. Wie aus Fig. 2 ersichtlich ist, sind die Eingänge 8 mit dem Logikbaustein 10 über Eingangsfilter 14 und die Ausgänge 9 mit dem Logikbaustein 10 über Ausgangstreiber 15 verbunden. Dadurch wird erreicht, daß der Logikbaustein 10 bei einer versehentlichen Fehlverdrahtung bzw. bei einem Kurzschluß oder ähnlichen Fehlfunktionen nicht beschädigt wird. Weiterhin ist durch die Eingangsfilter 14 ein Entprellen der Eingangssignale möglich. Auch können über die Filter 14 und die Treiber 15 Signalpegelanpassungen vorgenommen werden, z. B. von 20 mA auf 5 V.

Der Logikbaustein 10 ist über den Bus 16 sowie die Steuerleitungen 17 mit dem Prozessor 11 und damit auch mit dem Prozessor 6 verbunden. Dadurch wird es möglich, die korrekte Funktion des Logikbausteins 10 — auch während des Betriebs — zu überwachen. Zur Überwachung des Logikbausteins 10 können die Werte korrespondierender Eingänge 8 und Ausgänge 9 gleichzeitig zur Verarbeitung im Logikbaustein 10 an den Prozessor 11 und weiter an den Prozessor 6 übermittelt werden. Gegebenenfalls können auch Zwischenzustände des Logikbausteins 10, z. B. ein Merker oder ein Zählerstand, an den Prozessor 6 gemeldet werden. Es können auch neue Steuerungsparameter, z. B. neue Zeitkonstanten, an den Logikbaustein 10 übertragen werden.

Die in Fig. 2 dargestellten Steuerleitungen 17 dienen beispielsweise der Übertragung eines Resetsignals, mit dem die internen Merker rücksetzbar sind, sowie der Meldung des Logikbausteins 10 an den Prozessor 6 über seinen derzeitigen Programmierzustand, also z. B. der Meldung "Programmierung Logikbaustein geändert". An dieser Stelle sei erwähnt, daß die Programmierung des Logikbausteins 10 nur dann geändert werden kann, wenn der Logikbaustein 10 inaktiv ist, d. h. wenn er nicht in die Steuerung eines Prozesses eingebunden ist. Wenn der Logikbaustein 10 aus mehreren unabhängig voneinander funktionsfähigen Teilen besteht, ist es auch möglich, daß nur der Teil, dessen Programmierung geändert wird, inaktiv ist.

Gemäß Fig. 3 weist der Logikbaustein 10 ein Eingangslatch 20 und ein Ausgangslatch 21 auf, die beispielsweise mit einem Takt von 1 MHz getaktet sind. An die Eingänge des Eingangslatches 20 sind die Eingänge 8 angeschlossen. An die Ausgänge des Ausgangslatches 21 sind die Ausgänge 9 angeschlossen. Zwischen den Latches 20, 21 findet die eigentliche parallele Verarbeitung der Signale statt. Hierzu wird, beispielsweise, im

Logikschalter 22 eine elementare logische Verknüpfung der Eingänge E0 und E1 durchgeführt, gegebenenfalls auch mit Zwischenergebnissen, wie über die Leitung 23 angedeutet ist.

Das Ergebnis des Logikschalters 22 kann weiter verarbeitet werden oder auch direkt einem der Ausgänge zugeleitet werden, im gegebenen Fall dem Ausgang A0. Die Logikschalter 22 können, wie oben erwähnt, elementare logische Verknüpfungen durchführen, z. B. VERGLEICHEN, UND, ODER, NICHT-UND, NICHT-ODER. Um weitere, kompliziertere Funktionen realisieren zu können, ist es von Vorteil, wenn der Logikbaustein 10 speichernde Elemente 24 aufweist, aus denen dann z. B. Zähler, Zeitgeber oder Flankenmerker aufgebaut werden können.

Fig. 4 zeigt einen bevorzugten konstruktiven Aufbau der Ein-/Ausgabe-Baugruppe 3. Wie aus Fig. 4 ersichtlich ist, ist die Baugruppe 3 eine gekapselte Flachbaugruppe, die mit einem modular aufgebauten Baugruppenträger 25 verbunden ist. Die Baugruppe 3 weist einen Schacht 26 für das z. B. in Fig. 1 dargestellte Anwendermodul 13' und eine Schnittstelle 27 zum Anschluß eines Programmiergeräts auf. Über das Anwendermodul 13' und die Schnittstelle 27 ist es möglich, den in der Baugruppe 3 enthaltenen Logikbaustein 10 direkt, d. h. nicht über den Prozessor 6, zu programmieren.

Weiterhin weist die Baugruppe 3 zwei Sub-D-Steckkontakte 28a, 28b, wobei die Kontakte 28a zum Anschluß von Sensoren und die Kontakte 28b zum Anschluß von Stellgliedern dienen.

Die Kernidee der vorliegenden Erfindung ist es, ein herkömmliches, sequentielles Anwenderprogramm für eine speicherprogrammierbare Steuerung soweit wie möglich auf die aus der Schütztechnik bekannten Strukturen abzubilden, d. h. die korrespondierenden Ein- und Ausgänge über Logikelemente direkt zu verdrahten. Hierzu werden die logischen Bedingungen eines in einer Programmiersprache für speicherprogrammierbare Steuerungen erzeugten Anwenderprogrammes in eine Verbindungstabelle konvertiert und in einem Datenfeld abgelegt. Diese Daten werden dann in den Logikbaustein 10 geladen und führen dort zu einer entsprechenden internen Verschaltung des Logikbausteins 10. Dabei ist es, wie in Fig. 1 gezeigt, möglich, mehrere dieser Logikbausteine 10, 10' seriell und/oder parallel miteinander zu verschalten. Der Programmablauf wird dadurch auf die Zentraleinheit 2 und die Baugruppen 3, 3' verteilt.

Durch die direkte Verdrahtung korrespondierender Ein- und Ausgänge miteinander entfällt für diese das bei konventionellen speicherprogrammierbaren Steuerungen benötigte Prozeßabbild. Weiterhin wird die speicherprogrammierbare Steuerung extrem schnell, die Zykluszeit geht tendenziell gegen Null. Auch wird das Alarmreaktionsverhalten reproduzierbarer, da die Alarmreaktionszeit besser eingehalten wird.

Im obenstehend beschriebenen Ausführungsbeispiel wurde der Logikbaustein in einem modular aufgebauten Automatisierungsgerät verwendet. Ebenso ist jedoch auch die Verwendung in einem allein betriebsfähigen Automatisierungsgerät möglich. In der Minimalversion dieses Automatisierungsgeräts weist das Automatisierungsgerät keinen Prozessor mehr, sondern nur noch den Logikbaustein auf, so daß das abzuarbeitende Programm vom Logikbaustein allein ausgeführt wird. Die Programmierung des Logikbausteins erfolgt in diesem Fall entweder über eine Schnittstelle zu einem Programmiergerät oder über ein Speichermodul, das vom Anwender programmiert wurde.

Die Logikbausteine 10, 10' sind im vorliegenden Fall umfeldprogrammierbare Logikfelder (FPGAs). Fig. 5 zeigt einen Ausschnitt aus der inneren Struktur eines solchen Logikfeldes. Die innere Struktur weist eine zweidimensionale Matrix von z. B. 12×12 Logikblöcken 31 auf. Diese Matrix ist von einem Ring von Ein-/Ausgabe-Blöcken umgeben. Sowohl dem Anfang als auch dem Ende jeder (waagrechten) Reihe sind je zwei Ein-/Ausgabe-Blöcke zugeordnet. Gleiches gilt für die (senkrechten) Spalten. Die Ein-/Ausgabe-Blöcke sind der Übersichtlichkeit halber nicht dargestellt. Weiterhin sind jeder Reihe von Logikblöcken 31 je zwei nicht unterbrechbare Verbindungen 32 und jeder Spalte drei Verbindungen 33, von denen zwei einmal in der Mitte der Spalte unterbrochen werden können, zugeordnet. Diese Anordnung von Logikblöcken 31 und Ein-/Ausgabe-Blöcken ist durchsetzt von einem Netz mit 13×13 Schaltmatrizen 34, wobei benachbarte Schaltmatrizen 34 über je fünf Kurzverbindungen 35 miteinander verbunden sind.

Die Logikblöcke 31 weisen gemäß Fig. 6 einen Kombinatorikblock 310 auf, der aus maximal 5 Eingangsvariablen 311, zwei Ausgangsvariablen 312 ermittelt. Weiterhin weist der Logikblock 31 zwei Flipflops 313, 314 auf, deren Eingangssignal entweder aus einer der Ausgangsvariablen 312 des Kombinatorikblocks 310 oder aus einer direkt über den Eingang 315 eingegebenen Variable besteht. Die Ausgangssignale der Flipflops 313, 314 können entweder in den Kombinatorikblock 310 zurückgeführt werden oder aber als eines der Ausgangssignale 316 des Logikblocks 31 ausgegeben werden. Der Logikblock 31 ist also dahingehend programmierbar, welche logische und/oder Speicherfunktion er ausführen soll.

Die beiden Ausgangsfunktionen des Logikblockes 31 sind im Prinzip unabhängig voneinander, werden im vorliegenden Fall jedoch stets gleich gewählt, da jeder der beiden Ausgänge 316 mit je zwei der vier nächsten Nachbarn eines Logikbausteins direkt verbindbar ist. Dadurch, daß die beiden Funktionen identisch sind, wird also erreicht, daß das Ausgangssignal jedes Logikblocks 31 seinen vier nächsten Nachbarn als Eingangssignal zur Verfügung gestellt werden kann. Die Topologie wird folglich strukturierter.

Weiterhin können die Ausgänge 316 mit den sie umgebenden Kurzverbindungen 35 sowie den sie umgebenden Langverbindungen 32, 33 verbunden werden. Auch sind an den Kreuzungspunkten zwischen den Langverbindungen 32, 33 untereinander sowie zwischen den Langverbindungen 32, 33 und den Kurzverbindungen 35 noch elektrische Verbindungen programmierbar.

Die Schaltmatrizen 34 sind ebenfalls programmierbar. Sie können eine Vielzahl der theoretisch denkbaren Verschaltungsmöglichkeiten realisieren, z. B. waagrechte und/oder senkrechte Durchverbindungen, Kontaktieren von waagrechten mit senkrechten Kurzverbindungen 35 und Aufteilen von einer Verbindung auf zwei oder drei.

Die Ein-/Ausgabe-Blöcke sind jeweils mit einem Anschlußpin des Chips verbunden und können wahlweise entweder ein Signal eingeben oder ausgeben, wobei dieses Signal wahlweise getaktet werden kann oder nicht.

Die Programmierung der Logikblöcke 31, der Schaltmatrizen 34 und der Ein-/Ausgabe-Blöcke ist jeweils lokal in diesen Elementen gespeichert, die hierzu einen kleinen statischen Speicher (SRAM) aufweisen.

Bezüglich weiteren Einzelheiten über umfeldpro-

grammierbare Logikbausteine wird auf Herstellerhandbücher verwiesen, z. B. auf Handbücher über die XC 3000 Logic Cell Array Familie von Xilinx.

Zum Programmieren derartiger Logikfelder existieren ASIC-Design-Tools, mittels derer die Logikfelder in der Struktur des Logikfeldes angepaßten Stromlaufanweisungen programmierbar sind. Bei diesen Anweisungen muß der ASIC-Designer jedoch viele ASIC-spezifische Randbedingungen beachten. Solche Randbedingungen sind beispielsweise Gatterlaufzeiten, der Signalpegel von ungenutzten Gattereingängen usw. Es ist offensichtlich, daß eine derartige Programmierung hardwarenah und hochkomplex ist. Sie ist nur von ausgesprochenen Experten handhabbar.

Für das Umsetzen der gewünschten Programmierung in interne Verschaltungen des Logikbausteins 10 existieren Programme. Die Laufzeit dieser Programme, d. h. die Umsetzung des gewünschten Gesamtverhaltens in eine interne Verschaltung des Logikfeldes, beträgt, insbesondere wegen der vielfältigen Verbindungsmöglichkeiten, etliche Minuten, Stunden, manchmal sogar Tage.

Obenstehend erwähnte Spezialkenntnisse sind dem Anwender von speicherprogrammierbaren Steuerungen nicht zumutbar, ebensowenig die extrem langen Laufzeiten der Umsetzungsprogramme. Der SPS-Anwender erwartet Laufzeiten im Sekunden-, höchstens Minutenbereich. Im folgenden wird daher anhand eines Beispiels ein Verfahren beschrieben, mittels dessen ein in einer dem SPS-Anwender vertrauten Programmiersprache vorgegebenes Gesamtverhalten schnell und einfach in eine interne Verbindung des Logikfeldes umgesetzt werden kann.

Das Beispiel ist der Aufgabensammlung Simatic S5 der Siemens AG, Bestell-Nr. E 80850-C 345-X-AL entnommen und wird anhand von Fig. 7 erläutert.

"Wegen Bauarbeiten muß der Verkehr auf einer Straße über eine Fahrspur geleitet werden. Da das Verkehrsaufkommen sehr hoch ist, wird eine Bedarfsampelanlage installiert. Beim Einschalten der Anlage zeigen beide Ampeln Rot. Wird ein Initiator betätigt, so schaltet die entsprechende Ampel nach 10 Sekunden auf Grün. Die Grünphase soll mindestens 20 Sekunden andauern, bevor durch eventuelle Betätigung des anderen Initiators beide Signallampen wieder Rot zeigen. Nach 10 Sekunden wird dann die andere Fahrspur auf Grün geschaltet. Liegt keine Meldung eines Initiators vor, so bleibt die Ampelanlage in ihrem jeweiligen Zustand. Das Ausschalten der Anlage soll nur nach der Grünphase einer Fahrspur möglich sein. Beim Einschalten der Steuerung muß der Grundzustand (M0) ohne Bedingung gesetzt werden."

Zur Umsetzung des Problems in eine SPS-Programmiersprache wird zunächst eine Umbenennung der Symbole vorgenommen, wie in der untenstehenden Tabelle angegeben.

Symbol	Operand	Kommentar
S0	E0	Schalter Ein (Schließer)
I1	E1	Initiator 1 (Schließer)
I2	E2	Initiator 2 (Schließer)
H1	A1	Grün
H2	A2	Grün
H3	A3	Rot
H4	A4	Rot
M0	M0	Grundzustand M0
M1	M1	Zustand 1
M2	M2	Zustand 2
M3	M3	Zustand 3
M4	M4	Zustand 4
M5	M5	Zustand 5
M6	M6	Zustand 6
M7	M7	Zustand 7
	T1	Zeit 10 Sekunden
	T2	Zeit 20 Sekunden
	KT 100.1	Zeit für Zähler 1
	KT 200.1	Zeit für Zähler 2

Die zugehörige Verschaltung stellt sich in der SPS-Programmiersprache EUP (= Funktionsplan), wie in Fig. 8 gezeigt dar. Diese Art der Programmierung kennt der SPS-Anwender, und sie ist ihm geläufig. Die Aufgabe besteht darin, das vorgegebene, in einer SPS-Programmiersprache formulierte Gesamtverhalten schnell und einfach in eine FPGA-Struktur umzusetzen, so daß der SPS-Anwender im Ergebnis in die Lage versetzt wird, den Logikbaustein 10 selbst zu programmieren.

Erreicht wird dies dadurch, daß das Programm, das das SPS-Anwenderprogramm in die zugehörige interne Verschaltung des Logikbausteins 10 umsetzt, die theoretisch mögliche Komplexität des Logikbausteins 10 von vorneherein nur zu einem kleinen Bruchteil ausnutzt. Dies geschieht dadurch, daß ein Teil der im Prinzip frei wählbaren Verbindungen, z. B. die interne Verschaltung der Schaltmatrizen 34, im Umsetzungsprogramm fest vorgegeben wird, also vom Ersteller des SPS-Anwenderprogramms nicht beeinflußbar ist. Konkret werden die Verschaltungen der Schaltmatrizen 34 jeder der dreizehn senkrechten Spalten derart vorgegeben, daß zum einen die oberste, die unterste sowie die mittleren drei der Schaltmatrizen 34 einer Spalte die waagerecht verlaufenden Kurzverbindungen 35.1 ... 1 durchverbunden und die anderen der Kurzverbindungen 35 vorerst noch nicht verbinden und zum anderen die übrigen Schaltmatrizen 34 nur die senkrechten der Kurzverbindungen 35.1 ... 1 durchverbinden und die waagrechten Kurzverbindungen 35 blockieren.

Es ergibt sich dadurch eine Struktur, wie sie in Fig. 9 dargestellt ist. Es werden Gruppen 36 gebildet, die je fünf untereinander angeordnete Logikblöcke 31 enthalten und die vorne und hinten jeweils von fünf sich über die Länge einer "Halbspalte" erstreckenden Kurzverbindungen 37 umgeben sind. Auf diese Gruppen 36 wird die zu realisierende Schaltung von Fig. 8 auf noch zu erläuternde Art und Weise abgebildet. Die beiden waagrechten mittleren Reihen von Logikblöcken 31 werden auf ebenfalls noch zu erläuternde Art und Weise zur Erzeugung von Taktsignalen genutzt.

Die so entstandenen Gruppen 36 weisen eine handliche Größe auf. Einerseits ist ihre Komplexität klein genug und daher überschaubar genug, um auf relativ einfache Art und Weise abschätzen zu können, ob ein Teil-

netzwerk der zu realisierenden Gesamtschaltung durch eine der Gruppen 36 realisierbar ist, andererseits sind die Gruppen aber auch groß genug, um die Gesamtschaltung von Fig. 8 nicht in zu kleine Teilnetzwerke zerstückeln zu müssen. Als Kriterium zur Auswahl der Teilnetzwerke dienen die zur Verfügung stehenden Verbindungsressourcen und die zur Verfügung stehende Logikkapazität der Gruppen 36. Jedes Teilnetzwerk wird derart bemessen, daß es folgende Kriterien erfüllt:

a) es weist maximal fünf Eingangssignale auf;
b) es weist maximal fünf Ausgangssignale auf;
c) es werden zur Realisierung des Teilnetzwerks maximal fünf der Logikblöcke 31 benötigt und
d) die Verdrahtung des Teilnetzwerks innerhalb der Gruppe 36 ist möglich.

Beginnend beim Oder-Gatter 81 in Fig. 8 erkennt man sofort, daß auch das Und-Gatter 82 im selben Logikblock 31 realisierbar ist, da auch die Zusammenfassung dieser beiden Funktionen erst eine kombinatorische Funktion mit drei Eingängen und einem Ausgang ergibt. Dem RS-Flipflop 83 dagegen wird ein eigener Logikblock 31 zugewiesen, da jeder der Logikblöcke 31 aufgrund einer (willkürlichen) Compilervorschrift nur entweder eine kombinatorische Funktion ausführen oder eine Speicherfunktion wahrnehmen darf. Das Teilnetzwerk 84 kann folglich in einer Gruppe 36 realisiert werden, da insgesamt nur vier Eingangssignale, ein Ausgangssignal und zwei Logikblöcke 31 benötigt werden; die Kapazität einer Gruppe 36 also nicht überschritten wird.

Aufgrund ähnlicher Überlegungen ist leicht ersichtlich, daß auch die Teilnetzwerke 85 bis 88 in je einer Gruppe realisierbar sind. Vom nächsten Netzwerk 89 muß jedoch das Teilnetzwerk 90 abgetrennt werden, da sonst die Zahl der Eingänge den maximal zulässigen Wert von fünf überschritte.

Analog werden die anderen Netzwerke 91 bis 100 der Gesamtschaltung aufgeteilt, aber noch nicht bestimmten Gruppen 36 zugeordnet.

Eine gewisse Schwierigkeit bei der Aufteilung der einzelnen Netzwerke bereitet die Realisierung der Zeitglieder 99 und 100, da einem Zeitglied in der "SPS-Welt" kein entsprechendes Gegenstück in der "FPGA-Welt" gegenübersteht.

Um dem SPS-Anwender dennoch die leichte Programmierung von Zeitgliedern zu ermöglichen, wird diese für speicherprogrammierbare Steuerungen (SPSen) oft benötigte Funktion dem Anwender als Funktionsmakro zur Verfügung gestellt.

Zur Compilerlaufzeit erkennt der Compiler, daß ein Funktionsmakro vorliegt und setzt diesen Makro in eine interne, innerhalb des Logikfeldes verschiebbare Standard-Verbindung um. Die interne Standard-Verbindung wurde dabei vorab vom Compilerhersteller bzw. vom ASIC-Designer bestimmt. Dadurch wird der Compiler nicht in nennenswertem Umfang mit der Ermittlung der Verbindungen belastet, die den Funktionsmakro realisieren.

Fig. 10 zeigt ein Beispiel einer solchen Standard-Verbindung für einen Zeitzähler, der bis zu 210 Taktzyklen abzählen kann. Die tatsächlich zählbare Zeit ist selbstverständlich noch von der Taktung des Zählers abhängig.

Das in Fig. 10 gezeigte Beispiel benötigt drei nebeneinanderliegende Gruppen 36 von Logikblöcken 31. Die genaue Abbildung der in Fig. 10 dargestellten Logik auf FPGA-Strukturen ist dabei für den SPS-Anwender irrelevant. Bei der Erstellung derartiger Hardmakros, die mit Standard-ASIC-Design-Tools erfolgt, muß der Compiler-Hersteller bzw. der ASIC-Designer jedoch darauf achten, daß nur lokale Verbindungen, also direkte Verbindungen und Kurzverbindungen 35 verwendet werden; nicht aber globale Langverbindungen 32, 33. Hierdurch sind diese Makros nicht nur innerhalb des Logikfeldes leicht verschiebbar, also relocierbar. Sie sind auch unabhängig von den sie umgebenden Netzwerken oder Makros placierbar.

Da Hardmakros dem SPS-Programmierer (oder Anwender) über eine Bibliothek zur Verfügung gestellt werden, die Makros also vorab erstellt worden sind, ist die interne Konfiguration eines solchen Makros auch nicht an die begrenzten Möglichkeiten der Anwenderprogrammierung gebunden, sondern es kann die volle Komplexität des beanspruchten Feldbereichs ausgenutzt werden. Die Restriktionen der Anwenderprogrammierung können entfallen.

Die Erstellung derartiger Hardmakros durch den Compiler-Hersteller bzw. den ASIC-Designer und auch der Lauf der Umsetzungsprogramme kann zwar Stunden oder sogar Tage dauern. Dies ist in diesem Fall aber möglich und tolerierbar. Zum einen sind nämlich nur 3 - 5 - 15 der Logikblöcke 31 statt 12 · 12 = 144 Logikblöcke 31 miteinander zu verschalten. Zum anderen werden die Makros, wie bereits erwähnt, vorab erstellt. Der Anwender wird folglich nicht mit der Erstellung dieser Makros belastet, sondern sie stehen ihm sofort zur Verfügung. Das Zuordnen des Hardmakros zu einer bestimmten Stelle im FPGA dauert aber nur Bruchteile von Sekunden. Beim Design des Hardmakros ist lediglich zu beachten, daß die vier Ein- bzw. Ausgänge "Start", "Reset", "Clock" und "Zeitablauf" leicht zugänglich sind.

Für andere mögliche Standard-Funktionen der "SPS-Welt" sind selbstverständlich gegebenenfalls auch größere oder kleinere dieser Hardmakros möglich.

Nach der Aufgliederung der Gesamtschaltung in Teilnetzwerke 84 bis 98 werden diese zusammengefaßt, soweit auch die Zusammenfassung die obenstehend beschriebenen Kriterien a) bis d) erfüllt. Es ergibt sich beispielsweise, daß die Teilnetzwerke 87 und 94 sowie die Teilnetzwerke 93 und 97 zusammenfaßbar sind. Dieser soeben beschriebene Schritt ist nicht unbedingt nötig, er erhöht aber den Ausnutzungsgrad des Logikfeldes.

Falls wider Erwarten im Einzelfall zur Realisierung der gewünschten Verschaltung die Zahl von fünf Eingängen bzw. fünf Ausgängen überschritten werden muß, kann dies dadurch realisiert werden, daß — je nach Bedarf — eine oder mehrere Gruppen 36 vor der Gruppe 36, die mehr als fünf Eingangssignale benötigt, freigelassen werden und diese Signale ausnahmsweise mittels der waagrechten Kurzverbindungen und/oder der direkten Verbindungen von Logikblöcken 31 der davorliegenden Gruppe 36 der Gruppe 36 zugeführt werden, die mehr als fünf Eingänge benötigt. Falls auch diese zusätzlichen Verbindungsmöglichkeiten nicht ausreichen, wird eine Fehlermeldung generiert "gewünschte Schaltung nicht generierbar, Verbindungsmöglichkeiten zu gering".

Die einzelnen Teilnetzwerke 84 bis 100 werden nunmehr den einzelnen Gruppen 36 zugeordnet, so wie in Fig. 11 dargestellt. An dieser Stelle sei erwähnt, daß die Zuordnung der Teilnetzwerke 84 bis 100 auf die einzelnen Gruppen 36 gemäß ihrer Reihenfolge vorgenommen wurde. Dies ist die einfachste Art und Weise, eine

Zuordnung vorzunehmen; es sind aber auch komplexere Lösungen denkbar, die bereits die Verbindungen der Teilnetzwerke 84 bis 100 untereinander berücksichtigen.

Die außenliegenden Gruppen 36 werden nicht belegt, da die außenliegenden verlängerten Kurzverbindungen 37 nicht zur Anbindung dieser Gruppen zur Verfügung stehen, sondern anderweitig benötigt werden. Diese anderweitige Verwendung wird später noch erläutert werden.

Ein Beispiel einer komplexeren Lösung bei der Zuordnung der Teilnetzwerke 84 bis 100 auf die einzelnen Gruppen 36 bestünde beispielsweise darin, das Netzwerk 98 in der äußersten rechten Gruppe 36 anzuordnen. Dieses Netzwerk hat nämlich als einzigen Ausgang den Prozeßausgang A4. Dieser Prozeßausgang aber könnte direkt auf einen Ein-/Ausgabe-Block gelegt werden. Es würden also weder verlängerte Kurzverbindungen 37 noch sonstige globale Verbindungsressourcen benötigt.

Nach der Zuordnung der Teilnetzwerke 84 bis 100 zu den Gruppen 36 werden die internen elektrischen Verbindungen festgelegt. Hierbei werden zunächst soweit als möglich die direkten Verbindungen zwischen den Logikblöcken 31 ausgenutzt. Im vorliegenden Beispiel der Ampelschaltung sind dies nur wenige, zumeist läßt sich nur der Merkerausgang der Teilnetzwerke 84 bis 91 in das jeweils nächste Netzwerk weiterverbinden. Selbst dies ist im vorliegenden Fall aber nicht sinnvoll, da die Ausgangssignale der einzelnen Teilnetzwerke auch anderweitig benötigt werden und daher in jedem Fall auf globalere Verbindungen zurückgegriffen werden muß.

Als erstes werden die Ein- und Ausgangssignale vom und zum zu steuernden Prozeß verbunden, also die Eingangssignale E0 bis E2 und die Ausgangssignale A1 bis A4. Soweit möglich, werden die Ein- und Ausgangssignale direkt über die waagrechten Langverbindungen 32 den äußersten der verlängerten Kurzverbindungen 37 zugeführt. Falls die waagrechten Langverbindungen 32 bereits belegt sind, z. B. weil drei Signale anzuschließen sind, aber nur zwei waagrechte Langverbindungen 32 zur Verfügung stehen, werden die Signale zunächst auf senkrechte Langverbindungen 33 oder auf senkrechte Kurzverbindungen 37 gelegt. Dann werden sie über eine einer anderen Reihe von Logikblöcken 31 zugeordneten Langverbindung 32 an den Rand des Logikfeldes geführt. An den Rändern des Logikfeldes werden die Ein- und Ausgangssignale mittels der verlängerten Kurzverbindungen 37 derart weiterverbunden, daß z. B. das logische Eingangssignal E0 an den physikalischen Prozeßeingang E0 angeschlossen ist.

Durch einfaches Abzählen der verbleibenden internen Ein- bzw. Ausgänge der einzelnen Netzwerke 84 bis 100 ergibt sich sodann, daß ausnahmslos stets die fünf verlängerten Kurzverbindungen 37 zwischen den Teilnetzwerken 84 bis 100 ausreichen, um die Ein- und Ausgänge der Teilnetzwerke 84 bis 100 senkrecht miteinander zu vernetzen. Falls im Einzelfall mehr als fünf Leitungen benötigt werden würden, würde zur vollständigen Verbindung auf die senkrechten Langverbindungen 33 zurückgegriffen werden, vorzugsweise zunächst auf die unterbrechbaren der Langverbindungen 33.

Durch ebensolches einfaches Abzählen ergibt sich weiterhin, daß nunmehr nur noch dreizehn verschiedene Signale innerhalb des Logikfeldes geführt werden müssen, nämlich die acht Merkersignale M0 bis M7, die zwei Timer-Signale T1 und T2 sowie 3 interne Signale vom Teilnetzwerk 90 zum Teilnetzwerk 89, vom Teilnetzwerk 93 zum Teilnetzwerk 99 und vom Teilnetzwerk 94 zum Teilnetzwerk 100.

Nunmehr ist jedoch offensichtlich, daß diese interne Verbindung leicht möglich ist. Es werden nämlich einfach nacheinander die internen Ausgangssignale in der Reihenfolge ihres Entstehens auf die beiden äußeren der drei mittleren Kurzverbindungsleisten 43 gelegt. Hierdurch sind zehn interne Signale innerhalb des gesamten Logikfeldes abgreifbar. Sie stehen damit überall als interne Eingangssignale zur Verfügung.

Die drei noch zu verbindenden internen Ausgangssignale werden auf drei der waagrechten Langverbindungen 32 gelegt, so daß sie ebenfalls abgegriffen werden können, wo sie benötigt werden. Falls eines dieser drei Signale in der oberen Reihe der Gruppen 36 als Ausgangssignal anfällt, jedoch in der unteren Reihe von Gruppen 36 benötigt wird, wird dieses Problem wie folgt gelöst: Das jeweilige interne Ausgangssignal wird auf eine der waagrechten Langverbindungen 32 in der oberen Hälfte des Logikfeldes gelegt, diese waagrechte Langverbindung 32 mit einer senkrechten Langverbindung 33 verbunden und die senkrechte Langverbindung 33 mit einer waagrechten Langverbindung 32 verbunden, die in der unteren Hälfte des Logikfeldes angeordnet ist. Hierdurch steht dieses Signal auch in der unteren Hälfte des Logikfeldes zur Verfügung.

In analoger Weise wird selbstverständlich verfahren, wenn ein internes Signal in der unteren Hälfte des Logikfeldes erzeugt, jedoch in der oberen Hälfte als Eingangssignal benötigt wird.

Darüber hinaus lassen sich bei vorausschauender Anordnung der Netzwerke 84 bis 100 innerhalb des Logikfeldes drei der internen Signale direkt verbinden, so daß für die dann noch verbleibenden zehn internen Signale die beiden äußeren der Kurzverbindungsleisten 43 ausreichen.

Die oben stehend erwähnten drei internen Signale fallen nämlich jeweils nur einmal als Ausgangssignal an, nämlich in den Teilnetzwerken 90, 93 und 94, und werden auch nur einmal als Eingangssignale benötigt, nämlich von den Teilnetzwerken 89, 99 und 100. Wenn also die Teilnetzwerke 90 und 89, 93 und 99 sowie 94 und 100 jeweils unmittelbar hintereinander abgeordnet werden, können diese Signale direkt über Nächste-Nachbar-Verbindungen der Logikblöcke 31 untereinander verbunden werden. Auch ist eine Verbindung über die zwischen den jeweiligen Teilnetzwerkpaaren liegende verlängerte Kurzverbindung 37 möglich. In beiden Fällen werden keine waagrechten Verbindungen 32, 41, 42, 43 benötigt. Diese Verbindungen stehen damit anderweitig zur Verfügung.

Für die Taktung der Zeitzähler 99, 100 werden innerhalb des Logikfeldes Systemtakte von 1 ms, 10 ms, 100 ms und 1 sec bereitgestellt. Dies geschieht auf folgende Art und Weise. Mittels ASIC-Design-Tools werden hierzu vom Compiler-Hersteller vorab Teilerstufen erstellt, die einen beliebigen, von außen eingekoppelten Systemtakt auf ein 1/10, 1/100 und 1/1000 seiner ursprünglichen Frequenz herunterteilen. Dieser Makro, im folgenden Teilermakro genannt, wird dabei derart erstellt, daß er nur die beiden mittleren, bisher ungenutzten Reihen von Logikblöcken 31 sowie die direkten Verbindungen zwischen diesen Logikblöcken benötigt. Dieser Teil der (System-) Programmierung des FPGAs steht fest und ändert sich nicht. Von außerhalb des Logikbausteins 10 wird über einen der Ein-Ausgabe-Puffer ein Takt von 1 ms direkt in diesen Teilermakro eingekoppelt.

Die vier FPGA-internen Systemtakte von 1, 10, 100 und 1000 ms werden beispielsweise je einer der vier waagrechten Langverbindungen 32 zugeteilt, die den beiden mittleren Reihen von Logikblöcken 31 zugeordnet sind. Diese vier Zeittakte stehen damit im ganzen Logikfeld bereit und können dementsprechend abgegriffen werden. Welcher der Systemtakte an die Zeitzähler 99, 100 angeschlossen wird, ergibt sich für den Compiler aus der Bezeichnung der Eingangsvariablen KT_xy, x bezeichnet nach allgemeiner Regelung die Zahl der zu zählenden Taktzyklen und y ist ein Code für die Zeiteinheit. 200.1 bedeutet also beispielsweise, daß 200 Zyklen des Taktes mit dem Code 1, d. h. 100 ms zu zählen sind. Im Ergebnis mißt der Zeitzähler 100 also 200 · 100 ms = 20 sec.

Zum ordnungsgemäßen Ablauf des Steuerungsprogramms müssen im Regelfall die Logikbausteine 10, 10' und die Zentraleinheit 2 auch während des Betriebs Daten miteinander austauschen. Es kann beispielsweise sein, daß die Parametrierung des Logikbausteins 10 während des Betriebs geändert werden soll. Weiterhin sollte die Zentraleinheit 2 zumindest zeitweise über den aktuellen Zustand des Logikbausteins 10 (bzw. 10') informiert werden. Der Prozessor 6 und die Logikbausteine 10, 10' sind jedoch nicht miteinander synchronisiert. Es stellt sich daher das Problem der Datenkonsistenz. Das Problem wird dadurch noch vergrößert, daß der Datenverkehr zwischen Prozessor 6 und Logikbausteinen 10, 10' seriell verläuft. Der serielle Datenverkehr ist nötig, da ansonsten zu viele Pins der Logikbausteine 10, 10' für den Datenverkehr mit dem Prozessor 6 benötigt würden.

Das Problem wird dadurch gelöst, daß dem Anwender weitere Funktionsmakros zur Verfügung gestellt werden. Diese Funktionsmakros realisieren Schieberegister, die der Zwischenspeicherung von Ein- oder Ausgabedaten dienen sowie Arbeitsspeicher. Dabei werden zunächst die neu einzugebenden Daten vom Prozessor 6 in die Schreibzwischenspeicher z. B. des Logikbausteins 10 eingeschrieben. Während dieser Zeit sind die in den Zwischenspeichern abgespeicherten Werte zwar im Logikbaustein 10 vorhanden, werden aber vorerst nicht verwendet, da sie vorerst noch nicht freigegeben wurden. Mittels eines eigenen Befehls werden sodann die neu in den Logikbaustein 10 eingeschriebenen Werte von den Zwischenspeichern in die Arbeitsspeicher übernommen. Gleichzeitig werden die aus dem Logikbaustein 10 auszulesenden Werte in andere, sogenannte Lesezwischenspeicher eingelesen. Sodann werden die Daten seriell aus diesen Lesezwischenspeichern in den Prozessor 6 ausgelesen.

Fig. 12 zeigt ein Beispiel eines solchen Datenzyklus. Im vorliegenden Fall werden zum Übertragen aller benötigten Signale fünf Leitungen benötigt. Dabei werden auf den Leitungen folgende Informationen übertragen:

Solange der Signalpegel der Leitung RW Null ist, können Daten in die Schreibzwischenspeicher geschrieben werden. Solange der Signalpegel der Leitung RW 1 ist, können Daten aus den Leszwischenspeichern gelesen werden. Alle Zwischenspeicher sind derart an die Leitung RW angeschlossen, daß sie auf die ansteigende Signalflanke der Leitung RW getriggert sind. Zum Triggerzeitpunkt werden zum einen die Daten aus den Schreibzwischenspeichern in die Arbeitsspeicher übernommen. Zum anderen werden Daten aus den Logikblöcken 31 in die dafür vorgesehenen Leserwischenspeicher übernommen.

Die Signale PA1 und PA2 sind Adreßsignale. Mittels der Adreßsignale PA1 und PA2 können maximal je drei Schreibzwischenspeicher und Lesezwischenspeicher adressiert werden. Der theoretisch mögliche vierte Speicher (2 Signale = 2^2 = 4 Adressierungsmöglichkeiten) darf nicht verwendet werden. Diese Pegel werden nämlich an den Logikbaustein 10 angelegt, wenn keine Daten eingelesen bzw. ausgelesen werden. Daher darf diese Adresse, z. B. die Doppelnull, nicht verwendet werden.

CLK ist ein Takt. Wenn CLK Ems ist, liest der jeweils angesprochene Zwischenspeicher ein neues Bit ein bzw. aus.

Data ist die Datenleitung, auf der die Information selbst übertragen wird. Im vorliegenden Beispiel werden (rein zufällig) lauter Einsen übertragen.

Es ergibt sich aufgrund einfacher Überlegung, daß zum Lesen bzw. Schreiben der Zwischenspeicher mindestens vier Leitungen benötigt werden, nämlich die Leitungen RW, CLK, Data sowie mindestens eine Adreßleitung. Diese vier Signale werden auf die mittlere, bisher ungenutzte der drei Kurzverbindungsleisten 43 gelegt. Dadurch stehen diese vier Signale quer über den gesamten Logikbaustein 10 zur Verfügung. Falls überhaupt keine Parameter ein- und auszulesen sind, steht auch die mittlere der drei Kurzverbindungsleisten 43 für die interne Verbindung der Gruppen 36 zur Verfügung.

Falls mehr als drei Lesezwischenspeicher bzw. Schreibzwischenspeicher zu adressieren sind, werden weitere Adreßsignale PA3, PA4 etc. an den Logikbaustein 10 angelegt. Diese zusätzlichen Adreßsignale werden im Regelfall auf zwei waagrechte Langverbindungen 32 gelegt, wobei die eine der Langverbindungen 32 in der oberen Hälfte des Logikfeldes und die andere in der unteren Hälfte des Logikfeldes angeordnet ist.

Es bedarf keiner Erwähnung, daß die Bildung von zusätzlichen Speichermakros Logikfeldkapazitäten beansprucht. Diese Logikfeldkapazitäten stehen selbstverständlich anderweitig nicht mehr zur Verfügung.

Die obenerwähnten Speichermakros sind ebenso wie die Timer vorab vom Compilerhersteller mit ASIC-Design-Tools erstellt worden. Dem Compilerhersteller ist dabei im Rahmen seines allgemeinen Fachwissens bekannt, wie Register und Schieberegister aufzubauen sind. Ebenso ist in der Elektronik allgemein bekannt, wie Schieberegister mittels Adreßleitungen anzusteuern sind, so daß nun jeweils eines angesprochen wird. Derartige Speicherkonfigurationen bedürfen daher im Rahmen der vorliegenden Erfindung keiner weiteren Erläuterung.

Damit sind nun alle wesentlichen Schritte zur schnellen und einfachen Umsetzung eines SPS-Programms in eine FPGA-Struktur bekannt. Die nunmehr bekannten, internen elektrischen Verbindungen und die nunmehr ebenfalls ermittelte Programmierung der einzelnen Logikblöcke 31 werden in an sich bekannter Weise dem Logikfeld eingeprägt, so daß es also das gewünschte Gesamtverhalten, hier die Ampelsteuerung, realisiert. Weiterhin erhält der Anwender eine Meldung über den Ausnutzungsgrad des Logikfeldes oder, falls die Realisierung nicht möglich ist, eine diesbezügliche Meldung sowie eine Information darüber, warum die Realisierung nicht möglich war, z. B. weil keine Verbindungsreserven mehr zur Verfügung standen.

Die im vorliegenden Fall als Ausführungsbeispiel gewählte Ampelsteuerung ist selbstverständlich nicht so zeitkritisch wie andere Steuerungsvorgänge. Sie wurde jedoch gewählt, da sich anhand dieses einfachen Bei-

spiels die prinzipielle Vorgehensweise einfach erläutern läßt.

Im Ergebnis ergibt sich damit ein umfeldprogrammierbares Logikfeld, das zwar bei weitem nicht optimal ausgenutzt ist, dessen Programmierung aber schnell und einfach und vor allem in einer dem SPS-Anwender vertrauten Art und Weise erfolgt.

Patentansprüche

1. Speicherprogrammierbare Steuerung, insbesondere für Verpackungs- und Etikettiermaschinen, mit mehreren Ein- (8) und Ausgängen (9) zum Anschließen von Prozeßführungselementen, z. B. Sensoren oder Stellgliedern, dadurch gekennzeichnet, daß die Steuerung mindestens einen Logikbaustein (10) mit einer internen Verschaltung aufweist, über die mindestens ein Ausgang (A0) mit seinem korrespondierenden Eingang (E0) verbunden ist.

2. Steuerung nach Anspruch 1, dadurch gekennzeichnet, daß der Logikbaustein (10) parallelarbeitend ausgebildet ist.

3. Steuerung nach Anspruch 1 oder 2, dadurch gekennzeichnet, daß die interne Verschaltung des Logikbausteins (10) programmierbar und insbesondere auch reprogrammierbar ist.

4. Steuerung nach Anspruch 1, 2 oder 3, dadurch gekennzeichnet, daß der Logikbaustein (10) einen — vorzugsweise statischen — Speicher (12) zum Speichern der Bedingungen aufweist, die seine interne Verschaltung festlegen.

5. Steuerung nach Anspruch 1, 2, 3 oder 4, dadurch gekennzeichnet, daß der Logikbaustein (10) ein umfeldprogrammierbares Logikfeld (EPGA) ist.

6. Steuerung nach Anspruch 1, 2, 3, 4 oder 5, dadurch gekennzeichnet, daß die Steuerung mindestens einen Prozessor (6) und einen mit dem Prozessor (6) verbundenen Bus (5) aufweist, wobei der Ein- (8) und der Ausgang (9) sowohl mit dem Logikbaustein (10) als auch mit dem Bus (5) — auch zugleich — verbindbar sind.

7. Steuerung nach Anspruch 6, dadurch gekennzeichnet, daß der Prozessor (6) über mindestens eine Steuerleitung (17) mit dem Logikbaustein (10) verbunden ist.

8. Steuerung nach einem der obigen Ansprüche, dadurch gekennzeichnet, daß sie mindestens einen Anwenderspeicher (13) aufweist, der die interne Verschaltung des Logikbausteins (10) festlegt.

9. Steuerung nach einem der obigen Ansprüche, dadurch gekennzeichnet, daß der Logikbaustein (10) mindestens einen Anschluß (17) für ein Taktsignal aufweist.

10. Steuerung nach einem der obigen Ansprüche, dadurch gekennzeichnet, daß der Eingang (8) über einen Eingangsfilter (14) und der Ausgang (9) über einen Ausgangstreiber (15) mit dem Logikbaustein (10) verbunden ist.

11. Steuerung nach einem der obigen Ansprüche, dadurch gekennzeichnet, daß die Steuerung modular aufgebaut ist und der Logikbaustein (10) in einer Ein-/Ausgabe-Baugruppe (3) angeordnet ist.

12. Steuerung nach Anspruch 8 und 11, dadurch gekennzeichnet, daß die Ein-/Ausgabe-Baugruppe (3) einen Steckplatz (26) für den Anwenderspeicher (13) aufweist.

13. Steuerung nach Anspruch 11 oder 12, dadurch gekennzeichnet, daß die Baugruppe (3) eine Schnittstelle (27) zum Anschluß eines Daten-Ein-/Ausgabegeräts, z. B. eines Programmiergeräts, aufweist.

14. Steuerung nach Anspruch 11, 12 oder 13, dadurch gekennzeichnet, daß die Baugruppe mehrpolige Steckkontakte (28a, 28b) zum Anschließen der Prozeßführungselemente aufweist.

15. Verfahren zum Betreiben einer speicherprogrammierbaren Steuerung nach einem oder mehreren der obigen Ansprüche, dadurch gekennzeichnet, daß mindestens ein Eingangssignal in einen Logikbaustein (10) eingelesen und dort verarbeitet wird, so daß vom Logikbaustein (10) ein mit dem Eingangssignal korrespondierendes Ausgangssignal ausgebbar ist.

16. Verfahren nach Anspruch 15, dadurch gekennzeichnet, daß das Einlesen, Verarbeiten und Ausgeben getaktet ist.

17. Verfahren nach Anspruch 15 oder 16, dadurch gekennzeichnet, daß das Eingangssignal gefiltert, insbesondere entprellt, wird.

18. Verfahren nach einem der Ansprüche 15 bis 17, dadurch gekennzeichnet, daß der Logikbaustein (10) mit einem Prozessor (6) über mindestens eine Leitung (17) Daten austauscht.

19. Verfahren nach einem der Ansprüche 15 bis 18, dadurch gekennzeichnet, daß zum Testen der Programmierung des Logikbausteins (10) zumindest der Ein- (8) und der Ausgang (9) mit einem Bus (5) verbunden werden.

20. Verfahren nach einem der Ansprüche 15 bis 19, dadurch gekennzeichnet, daß der Datenverkehr des Logikbausteins (10) mit dem Bus (5) über einen dem Logikbaustein (10) zugeordneten Prozessor (11) erfolgt.

21. Programmierverfahren zum rechnergestützten internen elektrischen Verbinden eines umfeldprogrammierbaren Logikfeldes, das aus einer mindestens zweidimensionalen Anordnung von Logikblöcken (31) besteht, die durch vom Anwender frei festlegbare interne elektrische Verbindungen miteinander und mit dem Umfeld verbindbar sind,

- wobei aus einem vorgegebenen funktionalen Gesamtverhalten, z. B. aufgrund eines vorgegebenen funktionalen Schaltplans, insbesondere eines Funktionsplans für eine speicherprogrammierbare Steuerung, interne elektrische Konfigurationen, also Verbindungen und gegebenenfalls auch Logikfunktionen, des Logikfeldes bestimmt werden, welche das vorgegebene funktionale Gesamtverhalten realisieren,
- wobei die so bestimmten internen elektrischen Konfigurationen, also Verbindungen und gegebenenfalls auch die Logikfunktionen, dem Logikfeld eingeprägt werden,
- und wobei unabhängig von dem vorgegebenen funktionalen Gesamtverhalten beim Festlegen der internen elektrischen Verbindungen ein Teil der im Prinzip frei festlegbaren internen elektrischen Verbindungen fest vorgegeben wird.

22. Verfahren nach Anspruch 21, dadurch gekennzeichnet, daß die Logikblöcke (31) durch den fest vorgegebenen Teil der internen elektrischen Verbindungen in Gruppen (36) aufgeteilt werden, die zumindest teilweise gleiche Konfigurationen aufweisen.

23. Verfahren nach Anspruch 22, dadurch gekennzeichnet, daß das vorgegebene funktionale Gesamtverhalten in Teilfunktionen (84 – 100) zerlegt wird, die zumindest teilweise in je einer Gruppe (36) von Logikblöcken (31) realisierbar sind.

24. Verfahren nach Anspruch 22 oder 23, dadurch gekennzeichnet, daß für Standard-Teilfunktionen (99, 100), insbesondere komplexe Standard-Teilfunktionen (99, 100), interne elektrische Standard-Konfigurationen, also Verbindungen und gegebenenfalls auch Logikfunktionen, vorgebbar sind.

25. Verfahren nach Anspruch 24, dadurch gekennzeichnet, daß die Standard-Teilfunktionen (99, 100) durch mehr als eine Gruppe (36) von Logikblöcke (31) realisierbar sind.

26. Verfahren nach Anspruch 23 oder 24, dadurch gekennzeichnet, daß Teilfunktionen (84 – 98) und Standard-Teilfunktionen (99, 100), die in einer Gruppe (36) von Logikblöcken (31) realisierbar sind, zusammengefaßt werden, sofern auch die Zusammenfassung in einer Gruppe (36) von Logikblöcken (31) realisierbar ist.

27. Verfahren nach einem der Ansprüche 23 bis 26, dadurch gekennzeichnet, daß

- die Teilfunktionen (84 – 98) und/oder die Standard-Teilfunktionen (99, 100) und/oder die Zusammenfassung den Gruppen (36) von Logikblöcken (31) zugeordnet werden,
- unter Berücksichtigung der fest vorgegebenen internen elektrischen Verbindungen die internen elektrischen Verbindungen ermittelt werden, die die Gruppen (36) von Logikblöcken (31) derart miteinander verschalten, daß das vorgegebene funktionale Gesamtverhalten realisiert wird und
- die so ermittelten internen elektrischen Verbindungen dem Logikfeld eingeprägt werden, vorzugsweise zusammen mit den fest vorgegebenen internen elektrischen Verbindungen.

28. Verfahren nach Anspruch 27, dadurch gekennzeichnet, daß bei einem Logikfeld mit langreichweitigen Langverbindungen (32, 33) und kurzreichweitigen Kurzverbindungen (35) die elektrischen Verbindungen zu Prozeßeingängen und Prozeßausgängen zumindest teilweise über die Langverbindungen (32, 33) erfolgen.

29. Verfahren nach Anspruch 27 oder 28, dadurch gekennzeichnet, daß bei einem Logikfeld mit langreichweitigen Langverbindungen (32, 33) und kurzreichweitigen Kurzverbindungen (35) die internen elektrischen Verbindungen soweit als möglich über die Kurzverbindungen (35) erfolgen und nur die über die Kurzverbindungen (35) nicht realisierbaren internen elektrischen Verbindungen über die Langverbindungen (32, 33) erfolgen.

30. Verfahren nach Anspruch 29, dadurch gekennzeichnet, daß die Langverbindungen (32, 33) teilweise unterbrechbar sind und daß die internen elektrischen Verbindungen erst dann über die nicht unterbrechbaren Langverbindungen (32, 33) erfolgen, wenn die internen elektrischen Verbindungen über die unterbrechbaren Langverbindungen (32, 33) nicht realisierbar sind.

31. Verfahren nach einem der Ansprüche 21 bis 30, dadurch gekennzeichnet, daß das funktionale Gesamtverhalten in einer Programmiersprache für speicherprogrammierbare Steuerungen vorgegeben wird.

32. Verfahren nach einem der Ansprüche 21 bis 31, dadurch gekennzeichnet, daß das funktionale Gesamtverhalten in einer graphischen Programmiersprache vorgegeben wird.

Hierzu 14 Seite(n) Zeichnungen

Nummer: DE 42 06 524 A1
Int. Cl.5: G 06 B 19/05
Offenlegungstag: 27. August 1992

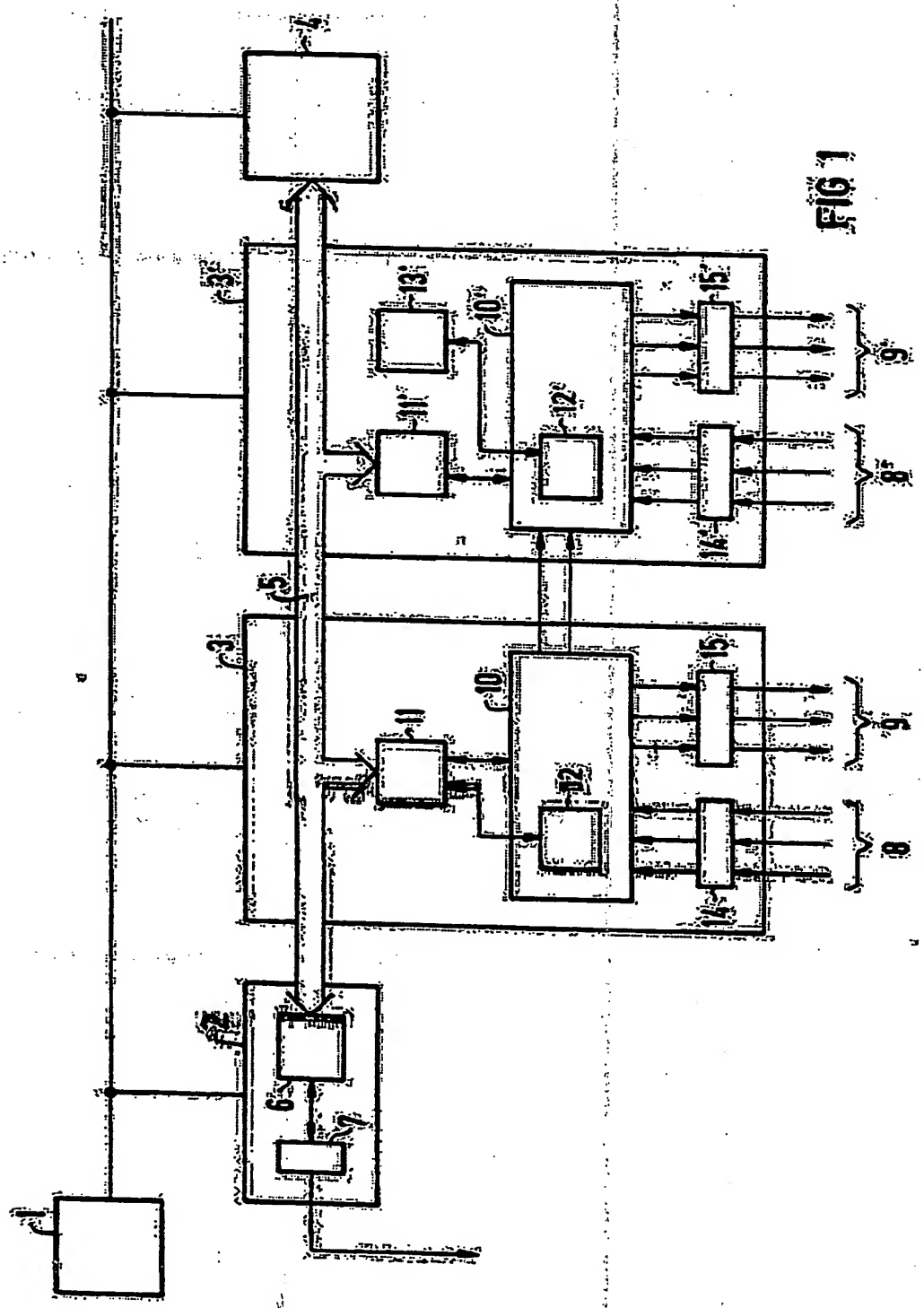


FIG 1

Nummer: DE 42 06 524 A1
Int. Cl.[5]: G 05 B 19/05
Offenlegungstag: 27. August 1992

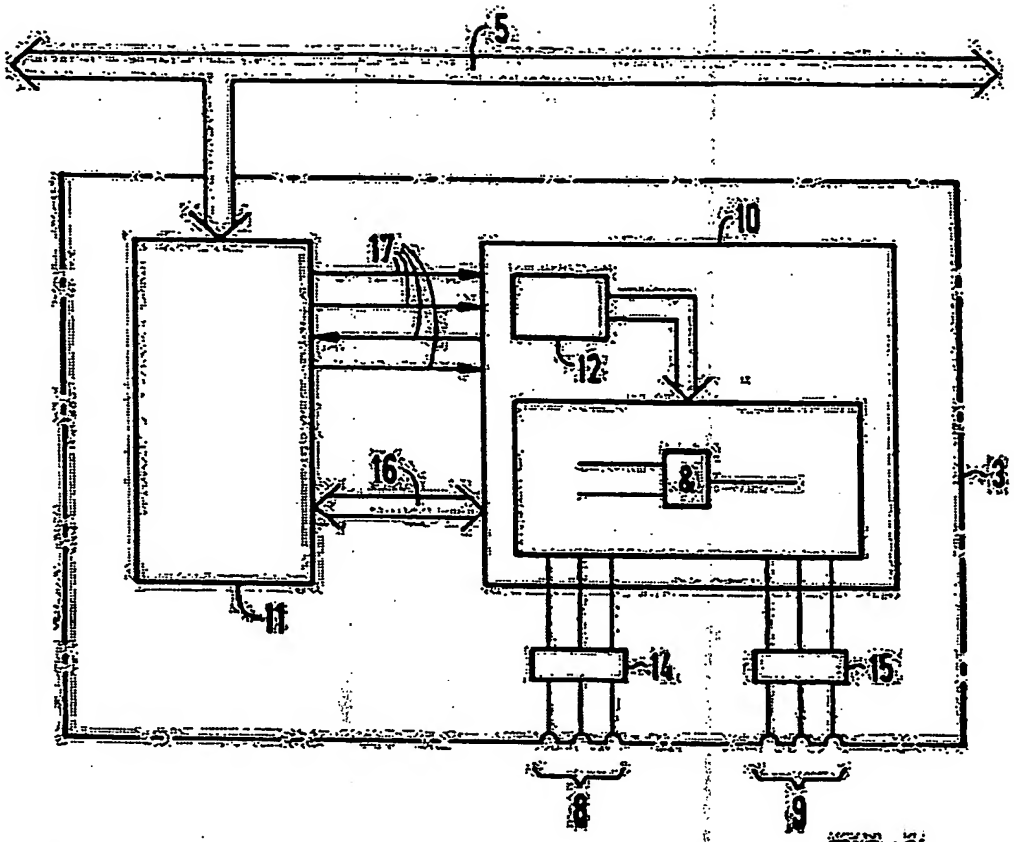


FIG 2

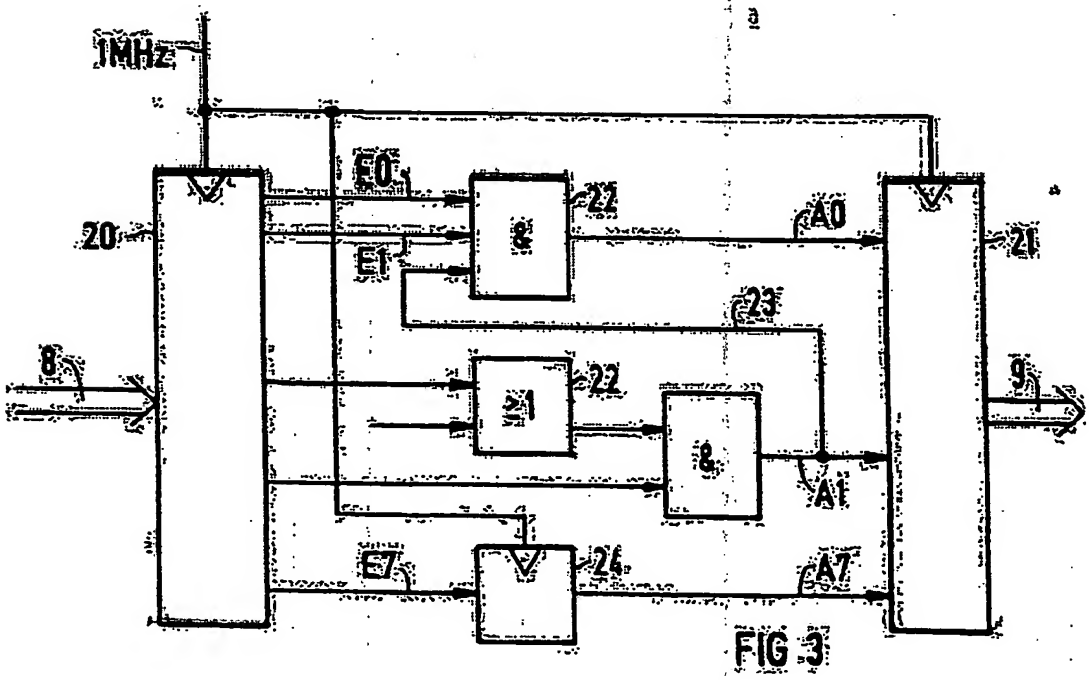


FIG 3

Nummer: DE 42 05 624 A1
Int. Cl.5: G 05 B 19/06
Offenlegungstag: 27. August 1992

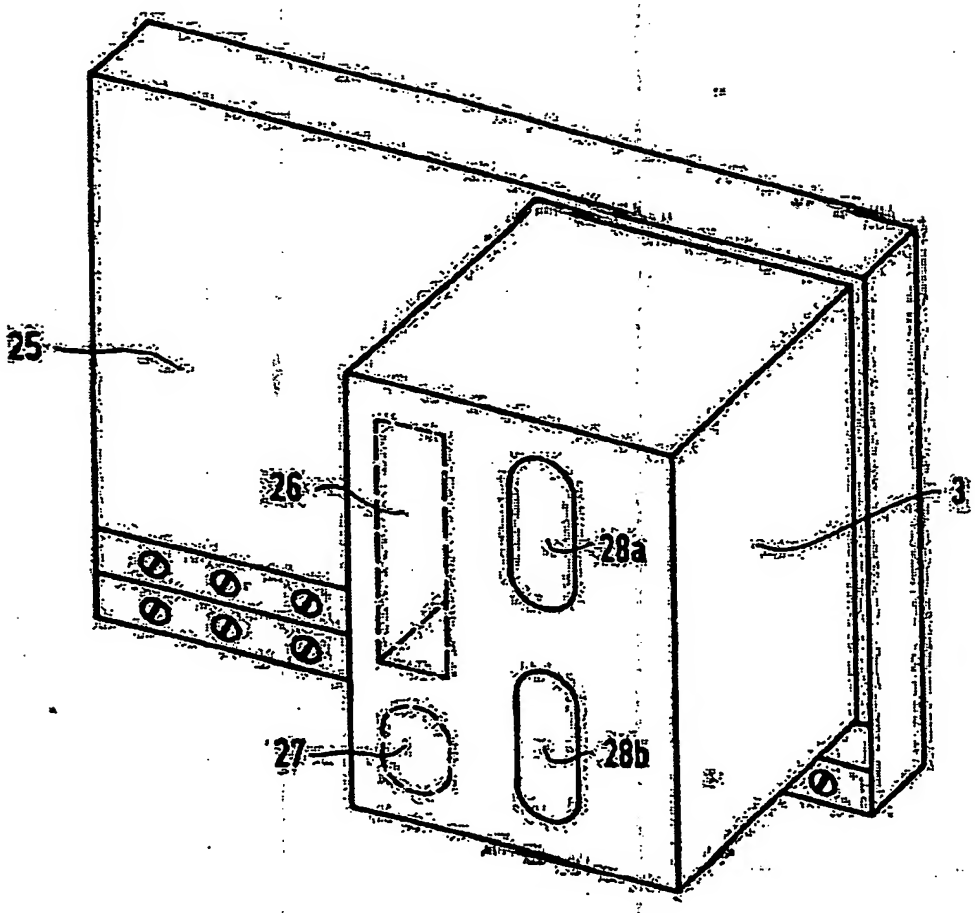


FIG 4

Nummer: DE 42 06 624 A1
Int. Cl.5: G 06 B 19/06
Offenlegungstag: 27. August 1992

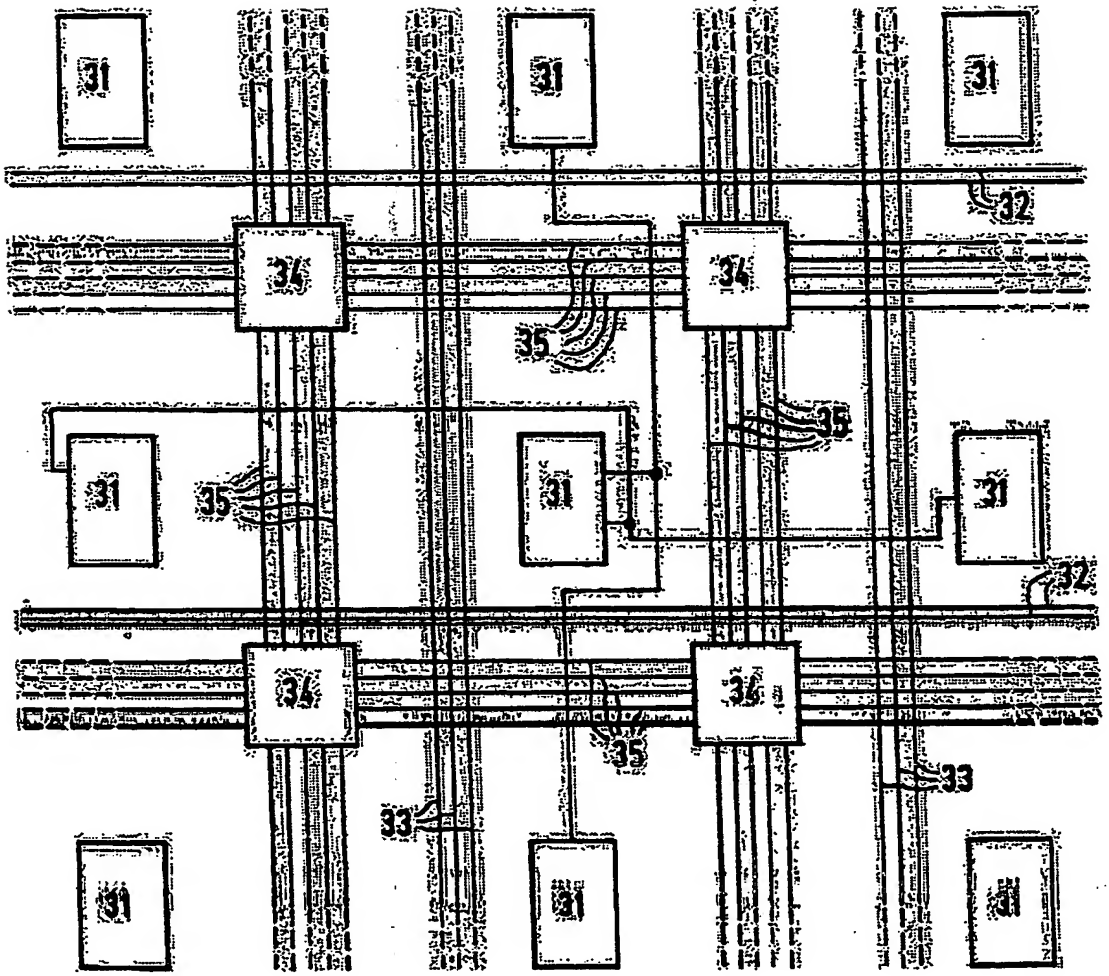


FIG 5

Nummer: DE 42 05 524 A1
Int. Cl.: G 06 B 19/06
Offenlegungstag: 27. August 1992

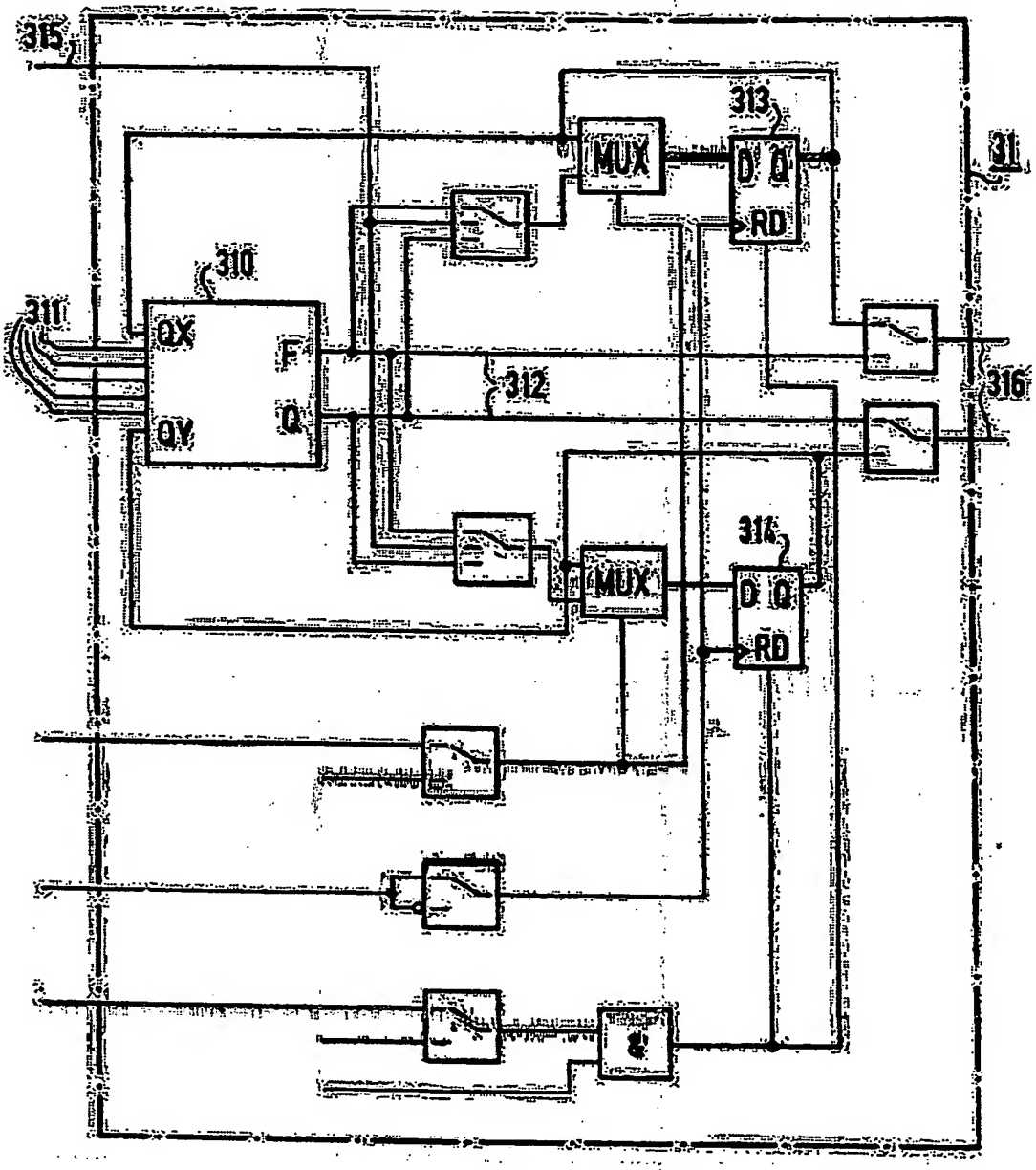


FIG 6

Nummer: DE 42 06 524 A1
Int. Cl.5: G 05 B 19/06
Offenlegungstag: 27. August 1992

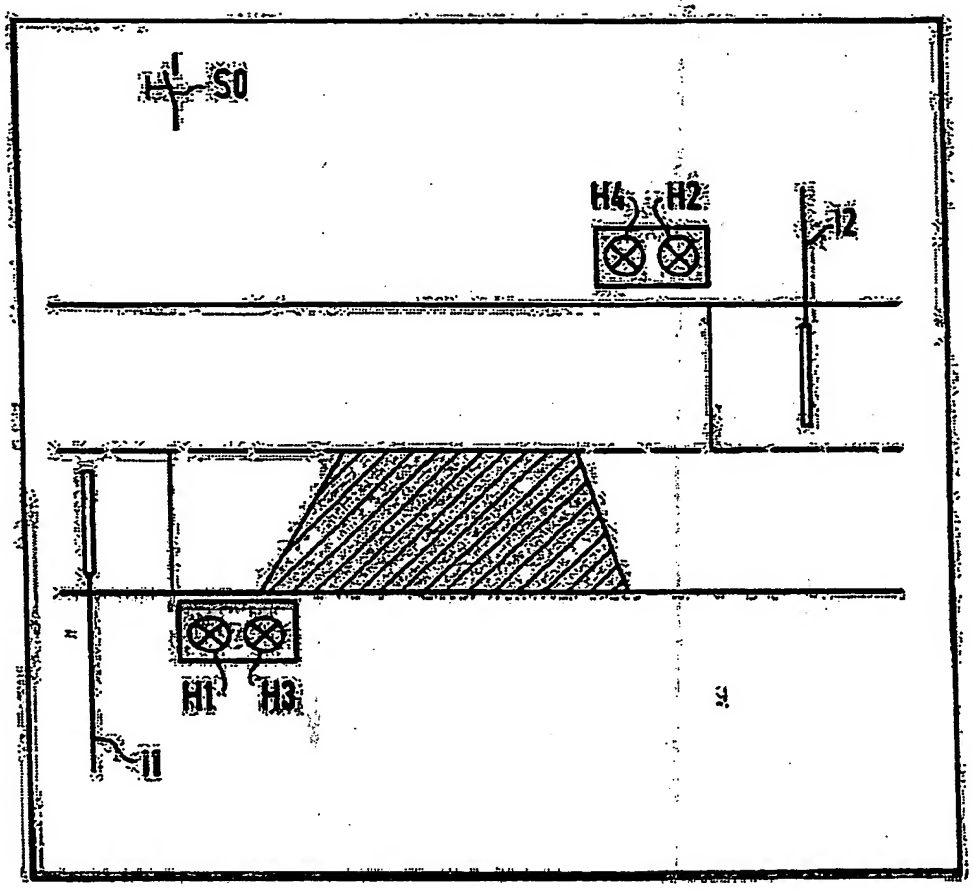


FIG 7

Nummer: DE 42 05 524 A1
Int. Cl.: G 05 B 19/05
Offenlegungstag: 27. August 1992

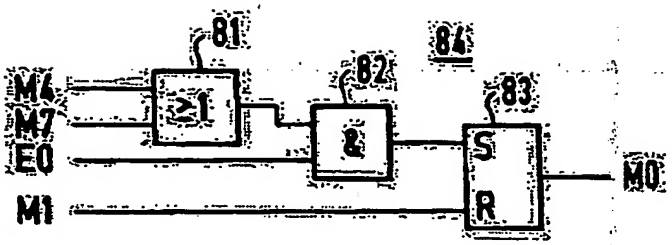


FIG 8a

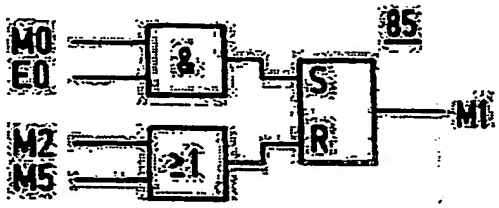


FIG 8b

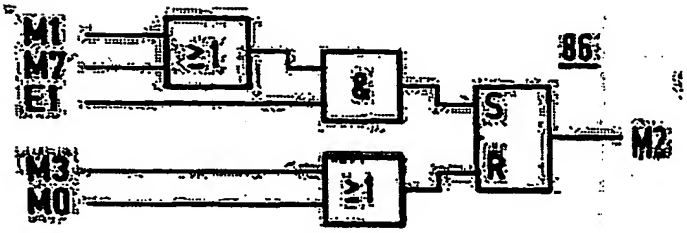


FIG 8c

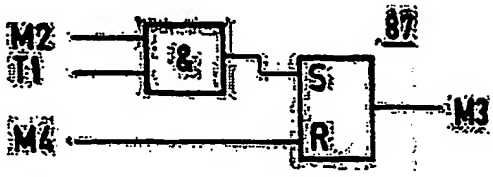


FIG 8d

Nummer: DE 42 06 524 A1
Int. Cl.5: G 05 B 19/05
Offenlegungstag: 27. August 1992

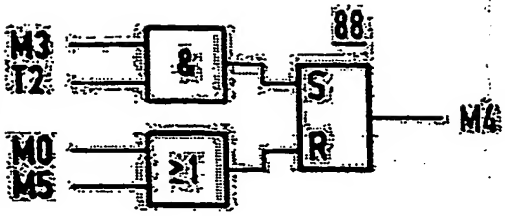


FIG 8e

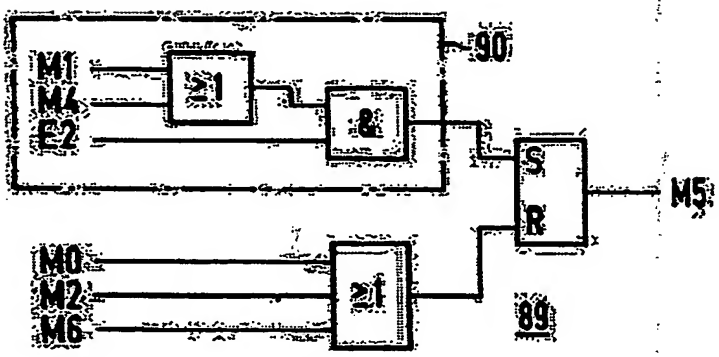


FIG 8f

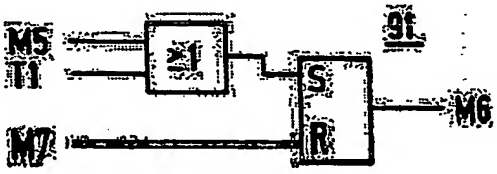


FIG 8g

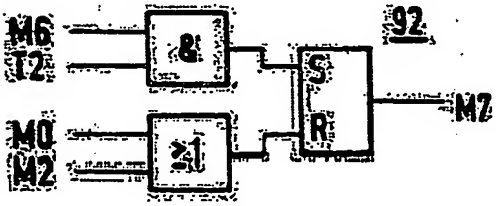


FIG 8h

(19) **FEDERAL REPUBLIC OF GERMANY**
GERMAN PATENT OFFICE
(12) **Offenlegungsschrift**
(10) **DE 42 05 524 A1**
(21) File reference: P 42 05 524.5
(22) Application date: 24.2.92
(43) Disclosure date: 27.8.92
(51) Int. Cl.[5]:
G 05 B 19/05
B 65 B 57/00
B 65 C 9/40

(30) Union priority: 32, 33, 31
14.11.91 EP 91 11 9483.5
(30) Internal priority: 32, 33, 31
22.02.91 DE 41 05 678.7
(71) Applicant:
Siemens AG, 8000 Munich DE

(72) Inventors:
Bock, Günther, Dipl.-Ing., 8450 Amberg, DE; Macht, Helmut, Dipl.-Ing., 8457 Kümmersbruck, DE; Wombacher, Christof, Dipl.-Ing. (FH), 8450 Amberg, DE; Prechtl, Manfred, Dipl.-Ing. (Univ.), 8470 Nabburg, DE; Lengemann, Andre, Dipl.-Ing. (FH), 8459 Edelsfeld, DE

(54) Stored program control

(57) A new stored program control, particularly for packaging and labelling machines, with several inputs (8) and outputs (9) for the connection of process control elements such as sensors or actuators, is proposed, which has at least a

logic module (10) with an internal interconnection via which at least an output is connected to its corresponding input (E0). A programming method for such a logic module (10) is also presented.

Description

The invention relates to a stored program control, particularly for packaging and labelling machines, with several inputs and outputs for connection of process control elements such as sensors or actuators, as well as a method for operating a stored program control and a method for computer-controlled internal electrical connection of a field programmable gate array.

Previously, machine controllers were built using protective technology. Protective circuits in fact work in parallel and are therefore fast, but they are prone to faults, complicated and are very difficult to build or adapt. The use of stored program controls has now become widespread. These work sequentially and are considerably easier to construct and program. But even modern stored program controls are often not fast enough, for example for controlling packaging and labelling machines, because of their sequential way of working. Controllers for these machines are as a rule also these days constructed on the basis of wired logic elements. This does give these controls a higher processing speed, but the wiring of the logic element is very complicated and fault-prone.

The object of the invention is to provide a stored program control, with which it is possible to deal with extremely fast control processes. Furthermore a method should be indicated that allows such a stored program control, containing a logic module with a field programmable gate array, to be rapidly and easily programmed.

The first object is achieved in that the controller has at least a logic module with an internal interconnection, via which at least an output is connected to its corresponding input. In this way, the output signal of this output is constantly adapted to the value of the corresponding input, so that an involved alarm reaction that would otherwise be necessary on this input can be dispensed with.

The logic module is advantageously designed to work in parallel so that several inputs and outputs can be connected together. This means that the output signals from these outputs are not subject to any fluctuations or variations caused by the processing time, but can be reproduced in a stable manner.

If the internal interconnection of the logic module is programmable and particularly also reprogrammable, the control can be easily adapted to changing requirements.

The programming of the logic module is quite simple here, if the logic module has a – preferably static – memory for storing the conditions that define its internal interconnection. The logic module can particularly be a field programmable gate array (FPGA).

If the control is of a modular design, the logic module is advantageously arranged in an input/output module, because then the system bus does not have to use the control for data transfer. In this case it is also an advantage if the logic module can be directly programmed on the module, e.g. via an interface for connection of a data input/output device, whereby the interface works directly or indirectly on the system bus or also on the logic module itself, or via a plug-in user memory, which defines the internal interconnection of the logic module.

The control works in such a way that at least an input signal is read into a logic module and is processed there, so that from the logic module an output signal corresponding to the input signal can be output.

The second object is achieved by the following method steps:

- from a predefined functional overall behaviour, e.g. on the basis of a predefined functional block diagram, particularly a control system flowchart for a stored program control, internal electrical configurations, and thus connections and if necessary also logic

functions, of the gate array are defined, which bring about the predefined functional overall behaviour, and
- the internal electrical configuration determined in this way, thus connections and if necessary logic functions, are impressed on the gate array,
- whereby independently of the predefined functional overall behaviour when defining the internal electrical connections a part of the, in principle, freely definable internal electrical connections has a fixed specification.

If the logic blocks are broken down into groups by the fixed specification part of the internal electrical connections, which at least in part have the same configurations, regular structures are generated, so that the gate array is broken down virtually into smaller units, namely the groups. In this way it is in fact possible to break down the predefined functional overall behaviour into sub-functions which at least in part can be performed in any group of logic blocks.

As a result, for standard sub-functions, particularly complex standard sub-functions, internal electrical standard configurations, and thus connections and if necessary also logic functions, can be specified, whereby in the individual case the standard sub-functions can also be implemented by more than one group of logic blocks.

If sub-functions and standard sub-functions in the individual case are very simple, if necessary several of these can be combined, provided that the combination into a group of logic blocks is feasible.

The programming of the gate array is then carried out in that

- the sub-functions and/or the standard sub-functions and/or the combination are assigned to the groups of logic blocks,
- taking into account the specified internal electrical connections the internal electrical connections are defined, the groups of logic blocks are interconnected to each other and to

external connections in such a way that the predefined functional overall behaviour is achieved, and

- the internal electrical connections defined in this way are impressed on the gate array, preferably along with the fixed specification internal electrical connections.

The programming of the gate array is particularly simple for the user, if the functional overall behaviour is specified in a programming language for stored program controls, particularly a graphical programming language.

Further advantages and details are evident from the following description of an embodiment, along with the drawings and in association with the other sub-claims. The drawings show as follows:

Fig. 1 – several modules of a modular design stored program control.

Fig. 2 – the internal structure of an input/output module;

Fig. 3 – the connections between inputs and outputs;

Fig. 4 – the structural design of an input/output module;

Fig. 5 – the internal structure of a field programmable gate array;

Fig. 6 – the design of a logic block;

Fig. 7 – an example of a problem to be solved in the form of a traffic light system;

Fig. 8 – the associated circuit engineering implementation of the traffic light system controller;

Fig. 9 – the advance definition of the internal electrical connections;

Fig. 10 – an example of an internal electrical standard connection;

Fig. 11 – the implementation of the specified overall behaviour in the field programmable gate array; and

Fig. 12 – communication between the logic modules and processor, in schematic form.

According to Fig. 1 a modular design stored program control comprises a power supply 1, a central unit 2, the input/output modules 3, 3' and other peripheral units 4. The modules 2, 3, 3', 4 are moreover connected together via a bus 5. The central unit 2 has at least a processor 6 for running a program and an interface 7 for data exchange with a programming device.

As can be seen further from Fig. 1, the module 3 has a logic module 10, which can, for example, be a field programmable gate array (FPGA). The logic module 10 is connected via the processor 11 with the bus 5 and thus also to the central unit 2. In this way it is possible, from a programming device, via the central unit 2, to program the internal interconnection of the logic module 10 in such a way that the inputs 8 are connected via the logic modules 10 according to logic conditions between the input and output signals derived in advance from the program to be run, to the outputs 9. The user generates for this with the abovementioned programming device two program parts: a time-uncritical part and a time-critical part. Both parts are transferred from the programming device to the processor 6 of the central unit 2. The time-uncritical part is stored in the central unit 2 and, as is usual with stored program controls, is run sequentially. The time-critical part is transferred on by the processor 6 to the logic modules 10, 10' and translated by these into a logic circuit.

The two program parts are completely independent of one another. Using special commands it is possible, however, for the processor 6 and the logic modules 10, 10' to exchange information.

Advantageously, moreover, the logic conditions which define the interconnection of the logic module 10, are transferred to a static memory 12 of the logic module 10 and the interconnection of the logic module 10 is defined on the basis of the content of the memory 12.

The module 31 likewise has a logic module 10 with a static memory 12', but the logic module 10' is programmed via a user memory 13'. If the interconnection of the logic module 10' has to be altered, the user memory 13' must be exchanged or reprogrammed, since the logic conditions, which define the interconnection of the logic module 10' are stored in the user memory 13'.

Fig. 2 shows, in a slightly different illustration, the electrical design of the module 3. As can be seen from Fig. 2, the inputs 8 are connected to the logic module 10 via input filters 14 and the outputs 9 to the logic module 10 via output drivers 5. In this way the logic module 10, in the event of an unintentionally incorrect interconnection or a short-circuit or similar malfunctions, is not damaged. Furthermore, via the input filters 14 debouncing of the input signals is possible. Also, via the filters 14 and the drivers 15 the signal level adaptations can be performed, e.g. from 20 mA to 5 V.

The logic module 10 is connected via the bus 16 and the control lines 17 to the processor 11 and thus also with the processor 6. In this way, it is possible to monitor the correct functioning of the logic module 10 – including during operation. In order to monitor the logic module 10 the values of corresponding inputs 8 and outputs 9 can simultaneously be transferred for processing in the logic module 10 to the processor 11 and further to the processor 6. If necessary intermediate states of the logic module 10, e.g. a marker or a

counter state, can also be reported to the processor 6. New control parameters, such as new time constants, can also be transferred to the logic module 10.

The control lines 17 shown in Fig. 2 are, for example, used to transmit a reset signal, with which the internal markers can be reset, and also for the reporting by the logic module 10 to the processor 6 of its current programming state, thus for example the message "Logic module programming changed". At this point mention is made of the fact that the programming of the logic module 10 can only be altered if the logic module is inactive, i.e. if it is not involved in the control of a process. If the logic module 10 consists of various independently functioning parts, it is also possible for only that part whose programming is being changed to be inactive.

According to Fig. 3, the logic module 10 has an input latch 20 and an output latch 21, which for example are clocked by a 1 MHz clock. The inputs 8 are connected to the inputs of the input latch 20. The outputs 9 are connected to the outputs of the output latch 21. Between the latches 20, 21 true parallel processing of the signals takes place. For this, for example, in the logic circuit 22 an elementary logic operation of inputs E0 and E1 is performed, if necessary also with intermediate results, as indicated by the line 23.

The result from the logic circuit 22 can be further processed or also supplied directly to one of the outputs, in the present case output A0. The logic circuits 22 can, as mentioned above, perform elementary logic operations such as, COMPARE, AND, OR, NOT-AND, NOT-OR. In order to be able to perform other more complicated functions, it is an advantage if the logic module 10 has memory elements 24, from which, for example, counters, timers or edge trigger flags can then be constructed.

Fig. 4 shows a preferred structural design of the input/output module 3. As can be seen from Fig. 4, module 3 is a encapsulated printed circuit board, which is connected to a modular design rack 25. The module 3 has a slot 26 for the user module 13', by way of example shown in Fig. 1, and an interface 27 for connecting a programming device. By

means of the user module 13' and the interface 27 it is possible to program the logic module 10 contained in the module 3 directly, i.e. not via the processor 6.

The module 3 also has two sub-D plug-in contacts 28a, 28b, whereby the contacts 28a are used for connecting sensors and the contacts 28b for connecting actuators.

The central idea of the present invention is to model a conventional, sequential user program for a stored program control, as far as possible on the structure known from protective technology, that is to say to wire the corresponding inputs and outputs via logic elements. For this the logic conditions of a user program generated in a programming language for stored program controls is converted into a net-list and stored in a data field. These data are then loaded into the logic module 10 and result there in a corresponding internal interconnection of the logic module 10. Moreover, it is possible, as shown in Fig. 1, to connect several of these logic modules 10, 10' to each other in series and/or in parallel. The program flow is thus distributed between the central unit 2 and the modules 3, 3'.

By means of the direct wiring together of corresponding inputs and outputs the process map that is necessary for conventional stored program controls can be dispensed with. Furthermore, the stored program control is extremely fast, with the "cycle time" tending towards zero. The alarm reaction behaviour is also reproducible, since adherence to the alarm response time is improved.

In the embodiment described above the logic module was used in a modular design automation device. The use in an automation device working independently is equally possible, however. In the minimal version of this automation device the automation device no longer has a processor, but just the logic module, so that the program to be run is executed by the logic module alone. Programming of the logic module takes place in this case either via an interface to a programming device or via a memory module, which has been programmed by the user.

The logic modules 10, 10' are in the present case field programmable gate arrays (FPGAs). Fig. 5 shows a section of the inner structure of such a gate array. The inner structure has a two-dimensional matrix of, for example, 12x12 logic blocks 31. This matrix is surrounded by a ring of input/output blocks. Both the start and the end of each (horizontal) row is assigned to two input/output blocks. The same applies to the (vertical) columns. The input/output blocks are not shown, for the sake of clarity. Furthermore, each row of logic blocks 31 is assigned two uninterruptible connections 32 and each column three connections 33, two of which can be interrupted once in the middle of the column. This arrangement of logic blocks 31 and input/output blocks is interspersed with a network of 13x13 switching matrices 34, whereby adjacent switching matrices 34 are connected to each other by means of five short connections 35.

The logic blocks 31 have, according to Fig. 6, a combinatorial block 310, which from a maximum of 5 input variables 311 determines two output variables 312. Furthermore, the logic block 31 has two flip-flops 313, 314, the input signal of which results either from one of the output variables 312 of the combinatorial block 310, or from a variable directly input via the input 315. The output signals of the flip-flops 313, 314 can either be fed into the combinatorial block 310 or output as one of the output signals 316 of the logic block 31. The logic block 31 can therefore be programmed for which logic and/or memory function it should execute.

The two output functions of the logic block 31 are in principle independent of each other, but in the present case are selected to always be the same, since each of the two outputs 316 can be directly connected to two of the four nearest neighbours to its logic module. Because the two functions are identical, the output signal of each logic block 31 can also have its four nearest neighbours made available as input signals. The result is a more structured topology.

Furthermore, the outputs 316 can be connected to the short connections 35 surrounding them and to the surrounding long connections 32, 33. At the points of intersection between the long connections 32, 33 themselves, and between the long connections 32, 33 and the short connections 35 further electrical connections can also be programmed.

The switching matrices 34 are likewise programmable. They can implement a large number of the theoretically conceivable switching possibilities, such as horizontal and/or vertical through-connections, contacting of horizontal and vertical cable connections 35 and splitting of one connection into two or three.

The input/output blocks are in each case connected to a connecting pin of the chip and can optionally input or output a signal, whereby the signal can be optionally clocked or not.

The programming of the logic blocks 31, the switching matrices 34 and the input/output blocks is in each case stored locally in these elements which have a small static random access memory (SRAM) for this.

With regard to further details of field programmable logic modules, reference is made to the manufacturers' manuals, such as the manuals for the XC 3000 Logic Cell Array Family from Xilinx.

In order to program such gate arrays ASIC design tools exist, by means of which the gate arrays can have their structure programmed with the current path instructions adapted to the gate array. With these instructions, however, the ASIC designer must observe a number of ASIC-specific constraints. Such constraints include, for example, gate times, the signal level of unused gate inputs, and so on. It is obvious that such programming is close to the hardware and highly complex. It can only be handled by true experts.

Programs exist for the translation of the desired programming into internal switchings of the logic module 10. The running time of these programs, i.e. the translation of the desired

overall behaviour into an internal interconnection of the gate array, is, particularly because of the numerous connection possibilities, a few minutes, hours, sometimes even days.

The abovementioned special knowledge is not reasonable for the user of stored program controls, even less so the extremely long running times of the translation programs. The SPC user expects running times of seconds, at the most a few minutes. In the following, therefore, using an example a method is described by which an overall behaviour specified in a programming language in which the SPC user is well versed can be quickly and easily translated into an internal connection of the gate array.

The example is taken from the Siemens AG Simatic S5 Collected Examples, order number E 80850-C 345-X-A1 and is explained using Fig. 7.

"As a result of construction work the traffic along a road has to use a single lane. Since the volume of traffic is very high, temporary traffic lights are installed. When the system is switched on, both sets of lights show red. When an initiator is pressed, the corresponding set of lights turns green after 10 seconds. The green phase should last at least 20 seconds, before through the possible operation of the other initiator both lights show red again. After 10 seconds the other lane is then on green. If no message is present at an initiator, then the light remains in its respective state. Switching off of the system should only be possible after a green phase for a lane. When the control is switched on the initial state (M0) must be set unconditionally".

In order to translate the problem into an SPC programming language initially a renaming of the symbols is carried out as indicated in the following table.

Symbol	Operand	Comment

S0	E0	Switch On (NO contact)
I1	E1	Initiator 1 (NO contact)
I2	E2	Initiator 2 (NO contact)
H1	A1	Green
H2	A2	Green
H3	A3	Red
H4	A4	Red
M0	M0	Initial state M0
M1	M1	State 1
M2	M2	State 2
M3	M3	State 3
M4	M4	State 4
M5	M5	State 5
M6	M6	State 6
M7	M7	State 7
	T1	Time 10 seconds
	T2	Time 20 seconds
	KT 100.1	Time for counter 1
	KT 200.1	Time for counter 2

The associated switching is given in the SPC programming language FUP (Function plan) as shown in Fig. 8. This type of programming is known to the SPC user and frequently used by him. The object is to translate the predefined overall behaviour formulated in an SPC programming language quickly and simply into an FPGA structure, so that the SPC user is ultimately put in a position where he can program the logic module 10 himself.

This is achieved in that the program that translates the SPC user program into the associated internal interconnection of the logic module 10, from the outset uses only a fraction of the theoretically possible complexity of the logic module 10. The way this happens is that part of the in principle freely selectable connections, e.g. the internal

interconnection of the switching matrices 34, has a fixed specification in the translation program, and can therefore not be influenced by the generator of the SPC user program. Specifically, the interconnections of the switching matrices 34 will specify each of the thirteen vertical columns in such a way that on the one hand the top, bottom and middle three switching matrices 34 of a column interconnect to the short connections 35 running horizontally 1:1 and the other short connections 35 for the time being do not connect, and on the other hand the other switching matrices 34 interconnect only the vertical short connections 35 1:1 and block the horizontal short connections 35.

This results in a structure, as shown in Fig. 9: groups 36 are formed, which each contain five logic blocks 31 arranged one below the other and which front and back are each surrounded by five short connections37 extending over the length of a "half-column". On these groups 36 the circuit from Fig. 8 to be created is modelled in a way and manner that will be explained further on. The two horizontal middle rows of logic blocks 31 are used, again in a way and manner to be explained further on, to generate clock signals.

The resultant groups 36 have a handy size: on the one hand their complexity is low enough and therefore clear enough to be able, in a relatively simple way and manner, to estimate if a sub-network of the overall switching to be created can be implemented by means of one of the groups 36, and on the other the groups are, however, large enough that the overall switching of Fig. 8 does not have to be cut down into sub-networks that are too small. A criterion for selection of the sub-networks, is the connection resources available and the logic capacity of the groups 36. Each sub-network is dimensioned so that it meets the following criteria:

a) it has a maximum of five input signals;
b) it has a maximum of five output signals;
c) a maximum of five logic blocks 31 are required to create the sub-network and
d) the sub-network can be wired within the group 36.

Starting with the OR-gate 81 in Fig. 8 it is immediately apparent that the AND-gate 82 can also be implemented in the same logic block 31, since the combining of these two functions only results in one combinatorial function with three inputs and one output. The RS flip-flop 83 on the other hand is assigned its own logic block 31, since eachof the logic blocks 31, because of a(n arbitrary) compiler specification should only either execute a combinatorial function or perform a memory function. The sub-network 84 can as a result be implemented in one group 36 since in total only four input signals, one output signal and two logic blocks 31 are needed, and the capacity of a group is therefore not exceeded.

Based on similar considerations it is easy to see also that the sub-networks 85 to 88 can each be implemented in one group. The sub-network 90 must, however, be separated from the next network 89, since otherwise the number of inputs exceeds the maximum permitted value of five.

Similarly the other networks 91 to 100 are split among the overall circuit, but are not yet assigned specific groups.

A certain difficulty arises in splitting the individual networks from the creation of the timing elements 99 and 100, since a timing element in the "SPC world" does not have a corresponding counterpart in the "FPGA world".

In order, despite this, to allow the SPC user to program timing elements easily, this function, which is frequently needed for stored program controls (SPCs) is made available to the user as a function macro.

From the compiler running time the compiler recognises that a function macro is present and translates this macro into an internal, standard connection that is movable within the gate array. The internal standard connection would have been described in advance here by the compiler manufacturer or by the ASIC designer. As a resultthe compiler is not to any

significant extent burdened with determining the connections, which implement the function macro.

Fig. 10 shows an example of such a standard connection for a timing element, which can count up to 210 clock cycles. The actual time that can be counted is of course further dependent on the clocking of the counter.

The example shown in Fig. 10 requires three groups 36 of logic blocks 31 alongside each other. Here, the precise modelling of the logic shown in Fig. 10 on FPGA structures is irrelevant for the SPC user. When generating such hard macros, which is performed using standard ASIC design tools, the compiler manufacturer or the ASIC designer must, however, ensure that only local connections, and thus direct connections and shot connections 35, are used, but not global long connections 32, 33. In this way, these macros are not only easy to move within the gate array, and thus relocatable, they can also be positioned independently of the networks or macros surrounding them.

Since hard macros are made available to the SPC programmer (or user) by means of a library, and the macros have therefore been created in advance, the internal configuration of such a macro is therefore also not bound by the limited user programming possibilities, but the full complexity of the gate array called upon can be used. User programming restrictions can be dispensed with.

The creation of such hard macros by the compiler manufacturer or the ASIC designer and also the running of the translation program can in fact take hours or even days. In this case, however, this is both possible and tolerable. For to begin with only 3x5 = 15 logic blocks 31 instead of 12x12 = 144 logic blocks 31 have to be connected to each other. And on the other hand the macros, as already mentioned, have been created in advance. As a result the user does not have to create these macros, but they are immediately available to him. The assignment of the hard macros to a certain point in the FPGA, however, lasts for only

fractions of seconds. When designing the hard macro all that has to be ensured is that the four inputs or outputs "start", "reset", "clock" and "timing" are easily accessible.

For other possible standard functions of the "SPC world" larger or smaller such hard macros are of course possible if necessary.

Following a breakdown of the overall circuit into sub-networks 84 to 98 these are combined, provided that the combination also meets the abovementioned criteria a) to d). It may be, for example, that the sub-networks 87 and 94 and sub-networks 93 and 97 can be combined. The step just described is not absolutely necessary, but it does increase the utilization of the gate array.

If unexpectedly in an individual case, in order to create the desired interconnection the number of five inputs or five outputs has to be exceeded, this can be implemented by –as required – leaving free one or more groups 36 before the group 36, requiring more than five input signals, and by way of exception feeding these signals through the horizontal short connections and/or the direct connections of logic blocks 31 of the frontmost group 36 to the group 36 that requires more than five inputs. If these additional connection possibilities are insufficient, an error message is generated "desired circuit cannot be generated, insufficient connection possibilities".

The individual sub-networks 84 to 100 are now assigned to the individual groups 36 as shown in Fig. 11. At this point it is mentioned that the assignment of the sub-networks 84 to 100 to the individual groups 36 was performed in sequence. This is the simplest way and manner in which to carry out any assignment; but more complicated solutions are also conceivable, which already take account of the connections between the sub-networks 84 to 100.

The external groups 36 are not occupied, since the external extended cable connections 37 are not available for connecting these groups, but are needed elsewhere. This use elsewhere will be explained further on.

An example of a more complex solution for the assignment of the sub-networks 84 to 100 to the individual groups 36 would, for example be to arrange the network 98 in the outer right group 36. For this network has as its sole output the process output A4. This process output, however, could be applied directly to an input/output block. Thus neither extended cable connections 37 nor other global connection resources would be needed.

Following the assignment of the sub-networks 84 to 100 to the groups 36 the internal electrical connections are defined. Here, initially, and as far as possible, the direct connections between the logic blocks 31 are used. In the present example of the traffic light circuit there are very few of these; usually only the marker outputs of the sub-networks 84 to 91 are interconnected to the respective adjacent network. Even this is not advisable in the present case, however, as the output signals of the individual sub-networks are also needed elsewhere and therefore in any case more global connections must be resorted to.

Firstly, the input and output signals from and to the process to be controlled are connected, thus the input signals E0 to E2 and the output signals A1 to A4. As far as possible, the input and output signals are fed directly via the horizontal long connections 32 of the most external of the extended cable connections 37. If the horizontal long connections 32 are already occupied, for example because three signals must be connected, but only two horizontal long connections 32 are available, the signals are initially applied to vertical long connections 33 or vertical short connections 37. Then they are fed via a long connection 32 assigned to another row of logic blocks 31 at the edge of the gate array. At the edges of the gate array the input and output signals are further connected by means of the extended short connections 37 in such a way that, for example the logic input signal E0, is connected to the physical process input E0.

By simply counting the remaining internal inputs or outputs of the individual networks 84 to 100, the result is then that, without exception, the five extended short connections 37 between the sub-networks,84 to 100 will always be sufficient to link the inputs and outputs of the sub-networks 84 to 100 vertically with one another. If in an individual case more than five lines are required, for complete connection the vertical long connections 33 would be resorted to, preferably initially the interruptible long connections 33.

Simple counting like this also means that now only thirteen more different signals have to be fed within the gate array, namely the eight marker signals M0 to M7, the two timer signals T1 and T2, and 3 internal signals from sub-network 90 to sub-network 89, from sub-network 93 to sub-network 99 and from sub-network 94 to sub-network 100.

It is now clear, however, that this internal connection is easily possible. For the internal outputs signals are simply applied one after another in the sequence in which there occur to the outer two of the three middle short connection strips 43. In this way, ten internal signals within the entire gate array can be picked off. They are thus available everywhere as internal input signals.

The three internal output signals remaining to be connected are applied to three of the horizontal long connections 32, so that they can likewise be picked off where they are needed. If one of these three signals occurs in the top row of the groups 36 as an output signal, but is needed in the bottom row of groups 36, this problem is resolved in the following way: the respective internal output signal is applied to one of the horizontal long connections 32 in the top half of the gate array, this horizontal connection 32 is connected to a vertical long connection 33 and the vertical long connection 33 is connected to a horizontal long connection 32, which is arranged in the bottom half of the gate array. In this way, the signal is also available in the bottom half of the gate array.

A similar method is obviously used if an internal signal is generated in the bottom half of the gate array, but is needed in the top half as an input signal.

Furthermore, in a forward-looking arrangement of the networks 84 to 100 within the gate array three of the internal signals can be directly connected, so that for the ten internal signals that then remain the outer two of the short connection strips 43 will suffice.

The abovementioned three internal signals each occur only once as an output signal, namely in the sub-networks 90, 93 and 94, and are also needed only once as input signals, namely by sub-networks 89, 99 and 100. If therefore the sub-networks 90 and 89, 93 and 99 and 94 and 100 are each assigned directly one behind the other, these signals can be connected together directly via nearest neighbour connections of the logic blocks 31. Connection via the extended short connection 37 located between the respective sub-network pairs is also possible. In both cases, no horizontal connections 32, 41, 42, 43 are needed. These connections are thus available elsewhere.

For the clocking of the timing elements 99, 100, within the gate array system clocks of 1 ms, 10 ms, 100 ms and 1 second are provided. This takes place in the following way and manner: by means of ASIC design tools the compiler producer will create in advance divider stages, which subdivide any external system clock connected into $1/10^{th}$, $1/100^{th}$ and $1/1{,}000^{th}$ of its original frequency. This macro, referred to as divider macro in the following, is created moreover in such a way that it needs only the two middle and thus far unused rows of logic blocks 31 and the direct connections between these logic blocks. This part of the (system) programming of the FPGAs is fixed and does not change. From outside the logic module 10 via one of the input-output buffers, a clock pulse of 1 ms is directly injected into this divider macro.

The four FPGA-internal system clock pulses of 1, 10, 100 and 1,000 ms are, for example, assigned one to each of the four horizontal long connections 32, which are assigned to the two middle rows of logic blocks 31. These four clocks are thus available throughout the gate array and can accordingly be picked off. The system clock that is connected to the timers 99, 100 is indicated to the compiler by the designation of the input variables KT x.y.

As a general rule x designates the number of clock cycles and y is code for the unit of time. 200.1 means therefore, by way of example, that 200 cycles of the clock with code 1, i.e. 100 ms, have to be counted. The result is that the timer 100 therefore measures 200 x 100 ms = 20 seconds.

For the correct operation of the control program, as a rule the logic modules 10, 10' and the central unit 2 must also exchange data during operation. It may, for example, be that the parameterisation of the logic module 10 has to be changed during operation. Furthermore, the central unit 2 should from time to time at least, be informed of the current state of the logic module 10 (or 10'). The processor 6 and the logic modules 10, 10' are not synchronised with each other, however. A problem therefore arises of data consistency. This problem is further magnified by the fact that data exchange between processor 6 and logic modules 10, 10' takes place serially. Serial data exchange is necessary since otherwise too many pins of the logic modules 10, 10' would be necessary for the data exchange with the processor 6.

The problem is solved by making available to the user further function macros. These function macros implement shift registers, which serve for intermediate storage of input or output data, as well as user memories. Here to begin with the new data to be input are written from the processor 6 into the write buffer store e.g. the logic module 10. During this time the values stored in the buffer stores are in fact available in the logic module 10, but are for the time being not used, since for the present they have not been enabled. By means of an intrinsic command the values newly written to the logic module 10 are transferred from the buffer memory to the user memory. At the same time the values to be read out from the logic module 10 are read into other, so-called read buffer stores. Then the data are read out serially from these read buffer stores into the processor 6.

Fig. 12 shows an example of such a data cycle. In the present case, five lines are needed for transfer of all signals needed. Here the following information is transferred on the lines:

as long as the signal level of the RW line is zero, data can be written to the write buffer stores. As long as the signal level of the RW line is 1, data can be read out from the read buffer stores. All buffer stores are connected to the RW line in such a way that they are triggered on the rising signal edge of the RW line. At the trigger point firstly the data are transferred from the write buffer stores into the user memory. Secondly the data are transferred from the logic blocks 31 to the read buffers stores provided for this purpose.

The signals PA1 and PA2 are address signals. By means of the address signals PA1 and PA2 a maximum of three write buffer stores and read buffer stores each can be addressed. The theoretically possible fourth store (2 signals = $2^2 = 4$ addressing possibilities) should not be used. For these levels are applied to the logic module 10, if no data are being read in or out. Therefore these addresses, e.g. the double zero, should not be used.

CLK is a clock. If CLK is one, the respective responding buffer store reads a new bit in or out.

Data is a data line, on which the information itself is transferred. In the present example (purely random) sheer ones are transferred.

Looked at simply, reading from or writing to the buffer store will require at least four lines, namely the RW, CLK and data lines and at least one address line. These four signals will be applied to the middle, so far unused, of the three short connection strips 43. In this way these four signals are available transversally across the entire logic module 10. If no parameters have to be read in or out anyway, the middle of the three short connection strips 43 is available for the internal connection of the groups 36.

If more than three read buffer stores or write buffer stores have to be addressed, further address signals PA3, PA4, etc., will be applied to the logic module 10. These additional address signals are as a rule applied to two horizontal long connections 32, whereby one of

the long connections is arranged in the top half of the gate array and the other in the bottom half of the gate array.

It goes without saying that the formation of additional memory macros calls for gate array capacities. These gate array capacities are of course no longer available elsewhere.

The abovementioned memory macros have, like the timers, been created in advance by the compiler producer using ASIC design tools. The compiler producer will know from his general technical knowledge how registers and shift registers are constructed. Equally, in the electronics world it will be generally known how to drive shift registers by means of address lines, so that only one is addressed in each case. Such memory configurations therefore require no further explanation in the context of the present invention.

So now all the essential steps for rapid and simple translation of an SPC program into an FPGA structure are known. The internal electrical connections that are now known and the likewise now identified programming of the individual logic blocks 31 are impressed in a known fashion on the gate array, so that it therefore provides the desired overall behaviour, here the traffic light control. Furthermore, the user receives a message concerning the level of use of the gate array or, if the implementation is not possible, a message to this effect and information on why the implementation was not possible, e.g. because no more spare connections were available.

The traffic light control selected in the present case as an embodiment is of course not as time-critical as other control processes. It was, however, selected since this example allowed a simple explanation of how to proceed.

The result is, therefore, a field programmable gate array that is a long way from being used to the optimum, but whose programming is fast and simple and above all is performed in a way and manner with which the SPC user is familiar.

Claims.

1. Stored program control, particularly for packaging and labelling machines, with several inputs (8) and outputs (9) for the connection of process control elements such as sensors or actuators, characterised in that the control has at least a logic module (10) with an internal interconnection via which at least an output (A0) is connected to its corresponding input (E0).

2. Control according to Claim 1, characterised in that the logic module (10) is designed for parallel operation.

3. Control according to Claim 1 or 2, characterised in that the internal interconnection of the logic module (1) is programmable and, particularly, also reprogrammable.

4. Control according to Claim 1, 2 or 3, characterised in that the logic module (10) has a – preferably static – memory (12) for storing the conditions that define its internal interconnection.

5. Control according to Claim 1, 2, 3, or 4, characterised in that the logic module (10) is a field programmable gate array (FPGA).

6. Control according to Claim 1, 2, 3, 4 or 5, characterised in that the control has at least a processor (6) and a bus (5) connected to the processor (6), whereby the input (8) and output (9) can be connected both to the logic module (10) and to the bus (5) – including simultaneously.

7. Control according to Claim 6, characterised in that the processor (6) is connected via at least a control line (17) to the logic module (10).

8. Control according to one of the above Claims, characterised in that it has at least a user memory (13'), which defines the internal interconnection of the logic module (10').

9. Control according to one of the above claims, characterised in that the logic module (10) has at least a connection (17) for a clock signal.

10. Control according to one of the above claims, characterised in that the input (8) is connected via an input filter (14) and the output (9) via an output driver (15) to the logic module (10).

11. Control according to one of the above claims, characterised in that the control has a modular design and the logic module is arranged in an input/output module (3).

12. Control according to Claim 8 and 11, characterised in that the input/output module (3) has a slot (26) for the user memory (13').

13. Control according to Claim 11 or 12, characterised in that the module (3) has an interface (27) for connection of a data input/output device such as a programming device.

14. Control according to Claim 11, 12 or 13, characterised in that the module has multi-pole contacts (28a, 28b) for connection of the process control elements.

15. Method for operating a stored program control according to one or more of the above claims, characterised in that at least an input signal is read into a logic module (10) and processed there, so that from the logic module (10) an output signal corresponding to the input signal can be output.

16. Method according to Claim 15, characterised in that the reading in, processing and output are clocked.

17. Method according to Claim 15 or 16, characterised in that the input signal is filtered and particularly debounced.

18. Method according to one of Claims 15 to 17, characterised in that the logic module (10) exchanges data with a processor (6) via at least a line (17).

19. Method according to one of Claims 15 to 18, characterised in that for testing of the programming of the logic module (10) at least the input (8) and output (9) are connected to a bus (5).

20. Method according to one of Claims 15 to 19, characterised in that the data exchange between the logic module (10) and the bus (5) takes place via a processor (11) assigned to the logic module (10).

21. Programming method for computer-controlled internal electrical connection of a field programmable gate array, which results from an at least two-dimensional arrangement of logic blocks (31), which can be connected to eachother via internal electrical connections that are freely definable by the user,
- whereby from a predefined functional overall behaviour, e.g. on the basis of a predefined functional block diagram, particularly a control system flowchart for a stored program control, internal electrical configurations, and thus connections and if necessary also logic functions, of the gate array are defined, which bring about the predefined functional overall behaviour,
- whereby the internal electrical configuration determined in this way, thus connections and if necessary logic functions, are impressed on the gate array,

- and whereby independently of the predefined functional overall behaviour when defining the internal electrical connections a part of the, in principle, freely definable internal electrical connections has a fixed specification.

22. Method according to Claim 21, characterised in that the logic blocks (31) are broken down into groups (36) by the fixed specification part of the internal electrical connections, which at least in part have the same configurations.

23. Method according to Claim 22, characterised in that the predefined functional overall behaviour is broken down into sub-functions (84 –100) which at least in part can each be performed in a group (36) of logic blocks (31).

24. Method according to Claim 22 or 23, characterised in that for standard sub-functions (99, 100), particularly complex standard sub-functions (99, 100), internal electrical standard configurations, and thus connections and if necessary also logic functions, can be specified.

25. Method according to Claim 24, characterised in that the standard sub-functions (99,100) can be implemented by more than one group (36) of logic blocks (31).

26. Method according to Claim 23 or 24, characterised in that the sub-functions (84-98) and standard sub-functions (99, 100), which can be implemented in a group (36) of logic blocks (31), are combined, provided that the combination into a group (36) of logic blocks (31) is feasible.

27. Method according to any of Claims 23 to 26, characterised in that
- the sub-functions (84 – 98) and/or the standard sub-functions (99, 100) and/or the combination are assigned the groups (36) of logic blocks (31),

- taking into account the specified internal electrical connections the internal electrical connections are defined, the groups (36) of logic blocks (31) are interconnected to each other in such a way that the predefined functional overall behaviour is achieved, and
- the internal electrical connections defined in this way are impressed on the gate array, preferably along with the fixed specification internal electrical connections.

28. Method according to Claim 27, characterised in that in a gate array with long trajectory long connections (32, 33) and short trajectory short comections (35) the electrical connections with process inputs and process outputs at least in part take place by means of the long connections (32,33).

29. Method according to Claim 27 or 28, characterised in that in a gate array with long trajectory long connections (32, 33) and short trajectory short connections (35) the internal electrical connections as far as possible are made via the short connections (35) and only the internal electrical connections that cannot be made via the short connections are made via the long connections (32, 33).

30. Method according to Claim 29, characterised in that the long connections (32, 33) are at least in part interruptible and in that the internal electrical connections are only then made via the uninterruptible long connections (32, 33), if the internal electrical connections cannot be made via the interruptible long connections (32, 33).

31. Method according to one of Claims 21 to 30, characterised in that the functional overall behaviour is specified in a programming language for stored program controls.

32. Method according to any of Claims 21 to 31, characterised in that the functional overall behaviour is specified in a graphical programming language.

---

14 page(s) of drawings follow

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- [x] BLACK BORDERS
- [ ] IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- [ ] FADED TEXT OR DRAWING
- [ ] BLURRED OR ILLEGIBLE TEXT OR DRAWING
- [ ] SKEWED/SLANTED IMAGES
- [ ] COLOR OR BLACK AND WHITE PHOTOGRAPHS
- [ ] GRAY SCALE DOCUMENTS
- [x] LINES OR MARKS ON ORIGINAL DOCUMENT
- [ ] REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- [x] OTHER: can't see clear

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.